VISIT…

From Technologies to Solutions

Building Telephony Systems with Asterisk

An easy introduction to using and configuring Asterisk to build feature-rich telephony systems for small and medium businesses

David Gomillion Barrie Dempster

PACKT PUBLISHING

Building Telephony Systems with Asterisk

An easy introduction to using and configuring Asterisk to build feature-rich telephony systems for small and medium businesses

David Gomillion
Barrie Dempster

BIRMINGHAM - MUMBAI

Building Telephony Systems with Asterisk

Copyright © 2005 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, Packt Publishing, nor its dealers or distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: September 2005

First reprint: February 2006

Production Reference: 1010206

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 1-904811-15-9

`www.packtpub.com`

Cover Design by `www.visionwt.com`

Credits

Authors
David Gomillion
Barrie Dempster

Reviewers
Rob Clews
Barrie Dempster
Alex Epshteyn
David Gomillion
Jan Kolasinski

Technical Editors
Richard Deeson
Niranjan Jahagirdar

Editorial Manager
Dipali Chittar

Development Editor
Louay Fatoohi

Indexer
Niranjan Jahagirdar

Proofreader
Chris Smith

Production Coordinator
Manjiri Nadkarni

Cover Designer
Helen Wood

About the Reviewers

Alex Epshteyn is the developer of Asterisk PBX Manager (a Webmin module for Asterisk) and the founding principal of Third Lane Technologies, LLC, a company specializing in VoIP software development and Asterisk consulting.

Rob Clews' first affair with a computer was with a Dragon 64. Since then he has become an avid developer and supporter of open-source software. Meeting Jan, he has founded Bluetel Solutions where he can stretch technologies to their limits and find the most efficient way to write and architect code to achieve the best results.

In what seems like a past life, **Jan Kolasinski** was a publisher for Wrox Press leading its Professional team. Since then he has been helping a number of small and medium sized companies apply technologies. In order to formalize this he has founded, with Rob, Bluetel Solutions where he tries to find new ways to help clients achieve better return on investments. Rob is the second reviewer.

About the Authors

Barrie Dempster was a Network Administrator/IT Manager for a growing call center when he saw the convergence and dependence of telephony and IT-related fields on each other. He focused on integration of telephony with IT infrastructure, and took on security as a career. The increase of voice-over-IP communications has now led to high demand for these skills, which he now utilizes in his current position as a Scotland-based Infrastructure and Security consultant for a variety of clients primarily within the financial sector.

He has been involved in varied projects, from building and deploying web and database servers to creating custom communication and conferencing systems, most of which are secured highly in order to survive public networks. He has deployed and used a variety of PBX systems and, as a strong supporter and user of free and open-source software, has a serious interest in Asterisk as it combines all of these interests into one extremely powerful package.

David Gomillion currently serves as Director of Information Technology for the Eye Center of North Florida. There, he orchestrates all of the technological undertakings of this four-location medical practice, including computers, software (off-the-shelf and custom development), server systems, telephony, networking, as well as specialized diagnostic and treatment systems.

David received a Bachelor's of Science in Computer Science from Brigham Young University in August, 2005. There he learned the theory behind his computer experience, and became a much more efficient programmer.

David has worked actively in the Information Technology sector since his freshman year at BYU. He has been a Networking Assistant, an Assistant Network Administrator, a Supervisor of a large Network and Server Operations unit, a Network Administrator, and finally a Director of Information Technology.

Through his increasing responsibilities, he has learned to prioritize needs and wants, and applies this ability to his Asterisk installations.

Table of Contents

Introduction 1

Chapter 1: Introduction to Asterisk 5

What is Asterisk? 5

Asterisk is a PBX 5

Station-To-Station Calls 6

Line Trunking 6

Telco Features 7

Advanced Call Distribution 7

Call Detail Records 7

Call Recording 8

Asterisk is an IVR System 8

Asterisk is a Voicemail System 8

Asterisk is a Voice over IP (VoIP) System 9

What Asterisk Isn't 11

Asterisk is Not an Off-the-Shelf Phone System 11

Asterisk is Not a SIP Proxy 11

Asterisk Does Not Run on Windows 12

Is Asterisk a Good Fit for Me? 12

Trade-Offs 12

Flexibility versus Ease of Use 12

Graphical versus Configuration File Management 13

Calculating Total Cost of Ownership 14

Return on Investment 15

Summary 15

Chapter 2: Making a Plan for Deployment 17

The Public Switched Telephony Network (PSTN) 17

Connection Methods 17

Plain Old Telephone Service (POTS) Line 17

Integrated Services Digital Network (ISDN) 18

T1 or E1 18

Voice over IP Connections 19

Determining Our Needs 20

Terminal Equipment **21**
Types of Terminal Devices 21
Hard Phones 21
Soft Phones 23
Communications Devices 24
Another PBX 25
Choosing a Device 25
Features, Features, and More Features… 25
Determining True Cost 26
Compatibility with Asterisk 27
Sound Quality Analysis 27
Usability Issues 28
Recording Decisions 28
How Much Hardware do I Need? **28**
Choosing the Extension Length **29**
Summary **32**

Chapter 3: Installing Asterisk **33**

Preparing to Install Asterisk **33**
Obtaining the Source Files **34**
Installing Zaptel **35**
Installing libpri **35**
Installing Asterisk **36**
Getting to Know Asterisk **39**
Summary **41**

Chapter 4: Configuring Asterisk **43**

Zaptel Interfaces **44**
zaptel.conf 44
Lines 45
Terminals 48
zapata.conf 48
Lines 52
Terminals 53
SIP Interfaces **54**
IAX Interfaces **59**
Voicemail **61**
Music On Hold **63**
Queues **64**

Conference Rooms **66**
Summary **67**

Chapter 5: Creating a Dialplan **69**

Creating a Context **69**
Creating an Extension **71**
Creating Outgoing Extensions **75**
Advanced Call Distribution **78**
Call Queues 78
Call Parking 82
Direct Inward Dialing (DID) 83
Automated Attendants **84**
System Services **87**
Summary **89**

Chapter 6: Quality Assurance **91**

Call Detail Records **91**
Flat-File CDR Logging 92
Database CDR Logging 93
Monitoring Calls **95**
Recording Calls **96**
Legal Concerns **97**
Summary **98**

Chapter 7: Asterisk@Home **99**

CentOS **99**
Preparation and Installation **100**
The Asterisk Management Portal (AMP) 101
Maintenance 103
Setup 104
Flash Operator Panel (FOP) 105
Flash Operator Configuration Files 106
Web MeetMe 106
Flexibility When Needed 106
A Simple One-to-One PBX **107**
Extensions 107
Trunks 108
Routes 108

Customer Relationship Management/SugarCRM **110**
Adding Contacts 111
Call Scheduling 111
Administration of SugarCRM 112
Configure Settings 112
User Management 112
User Roles 113
Summary **114**

Chapter 8: Case Studies **115**

Small Office/Home Office **115**
The Scenario 115
The Discussion 116
The Configuration 116
zaptel.conf 116
zapata.conf 117
musiconhold.conf 117
voicemail.conf 118
modules.conf 118
extensions.conf 119
Conclusions 119
Small Business **120**
The Scenario 120
The Discussion 120
The Configuration 121
zaptel.conf 121
zapata.conf 121
musiconhold.conf 122
agents.conf 122
queues.conf 122
sip.conf 123
meetme.conf 124
voicemail.conf 124
extensions.conf 125
Conclusions 128
Hosted PBX **128**
The Scenario 129
The Discussion 129

The Configuration 129
zaptel.conf 129
zapata.conf 130
musiconhold.conf 130
sip.conf 130
voicemail.conf 131
extensions.conf 131
Conclusions 134
Summary 134

Chapter 9: Maintenance and Security 135

Backup and System Maintenance 135
Backing Up Configurations 136
Backing Up Voice Data 138
Backing Up Log Files 139
Backup Scripts 139
Time Synchronization 142
Adding It All to cron 142
Rebuilding and Restoring the Asterisk Server 143
Disaster Recovery Plan (DRP) 143
Asterisk Server Security 144
Internal Access Control 144
Host Security Hardening for Asterisk 147
Integrity Checker 147
Root-Kit Detection 147
Automated Hardening 148
Role Based Access Control (RBAC) 148
Network Security for Asterisk 149
Firewalling the Asterisk Protocols 149
SIP (Session Initiation Protocol) 150
H.323 150
IAX 151
RTP—The Real-Time Transport Protocol 151
Controlling Administration of Asterisk 151
Asterisk Scalability 152
Load Balancing with DNS 153

Support Channels for Asterisk **154**
Mailing Lists 155
Forums 155
IRC (Internet Relay Chat) 155
Digium 156
Summary **156**

Index **157**

[PACKT PUBLISHING]

Introduction

Telephony systems are an integral part of business, and it's important that the framework used is flexible enough to cover as many areas of application as possible, and at the same time is user friendly. This book is an attempt at exploring one such system—Asterisk.

What This Book Covers

Chapter 1 introduces Asterisk and talks about the possible scenarios that would demand its usage, and the realistic trade-offs that you should consider when choosing it.

Chapter 2 discusses a basic deployment plan, and takes you through various aspects such as requirements and the how-tos of choosing the right terminal equipment and hardware.

Chapter 3 discusses installation of Asterisk. It starts with a section on preparing a system for installation, takes you through installation of necessary components, and ends with an introduction to the way Asterisk behaves.

Chapter 4 deals with the basic Asterisk configuration, and discusses the Zaptel interfaces in detail, and then the configuration of protocols and various features.

Chapter 5 deals with creating a dialplan. This involves creating a context and extensions, and the chapter also discusses advanced call distribution and automatic attendants.

Chapter 6 discusses quality assurance issues that concern most companies, and gives an overview of call detail records, call monitoring, and recording, etc.

Chapter 7 talks about Asterisk@Home—a simplified Asterisk solution that retains most of its functionality for its so-called "home" users—and a customer relationship management system, SugarCRM.

In *Chapter 8* we've shown a few case studies of working Asterisk-based phone systems, and have discussed scenarios for home offices and small businesses.

Chapter 9 deals with Asterisk's maintenance and security aspects. The topics range from backups of configuration files to disaster management plans to server security. This chapter also discusses Asterisk's scalability aspects and support channels.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

There are three styles for code. Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code will be set as follows:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Any command-line input and output is written as follows:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "clicking the Next button moves you to the next screen".

> Warnings or important notes appear in a box like this.

> Tips and tricks appear like this.

Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to `feedback@packtpub.com`, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the SUGGEST A TITLE form on `www.packtpub.com` or email `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the Submit Errata link, and entering the details of your errata. Once your errata have been verified, your submission will be accepted and the errata added to the list of existing errata. The existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with some aspect of the book, and we will do our best to address it.

PACKT
PUBLISHING

[PACKT PUBLISHING]

1
Introduction to Asterisk

In this chapter, we will be looking at what Asterisk is, and what it can do for us. As we explore features, we can make note of what features will help us to accomplish our goals.

What is Asterisk?

This is a fascinating question: what exactly is Asterisk? There are a number of answers, all of which are accurate.

First, Asterisk is a symbol (*). The symbol represents a wildcard in many computer languages. This gives us insight into the developers' hopes for Asterisk. It is designed to be flexible enough to meet any need in the telephony realm.

Second, Asterisk is open-source software. This means that hundreds, if not thousands, of developers are working every day on Asterisk, extensions of Asterisk, software for Asterisk, and customized installations of Asterisk. A big portion of the product's flexibility comes from the availability of the source code, which means we can modify the behavior of Asterisk to meet our needs.

Finally, and most importantly, Asterisk is a framework that allows selection and removal of particular modules, allowing us to create a custom phone system. Asterisk's well-thought-out architecture gives flexibility by allowing us to create custom modules that extend our phone system, or even serve as drop-in replacements for the default modules.

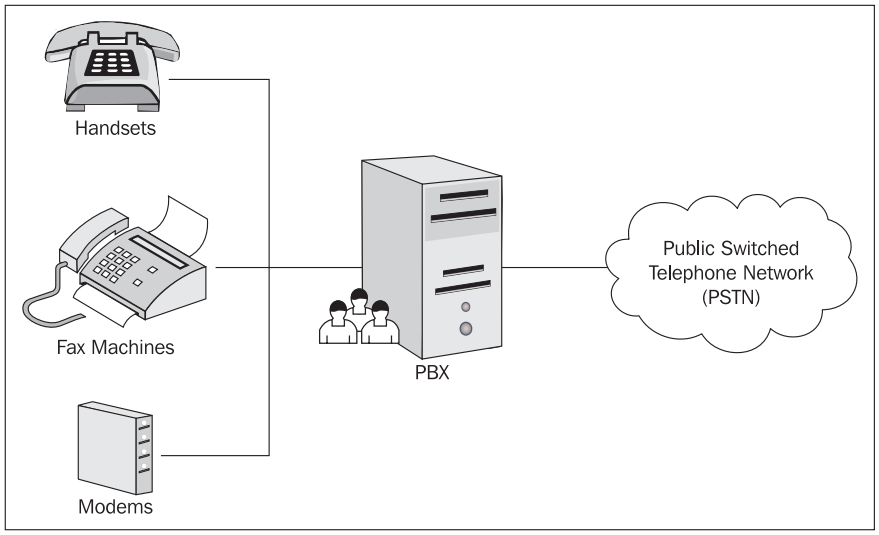
Asterisk is a PBX

Asterisk is a Private Branch Exchange (PBX). A PBX can be thought of as a private phone switchboard, connecting to one or more telephones on one side, and usually connecting to one or more telephone lines on the other. This is usually more cost effective than leasing a telephone line for each telephone needed in a business.

Station-To-Station Calls

First, as a PBX, Asterisk offers station-to-station calls. This means that users can dial from one phone to another phone. While this seems obvious, elementary phone systems are available (often referred to as Key Systems) that support multiple phones and multiple lines, and allow each phone to use any line. In operation, the handsets do not have individual extensions that can be dialed, and so there is no way to initiate a call from one handset to another. These systems can usually be identified by having all outgoing lines on every telephone, usually with a blinking light. Unlike Key Systems, Asterisk allows for station-to-station calls, allowing directed internal communications.

Consider for a moment the following diagram:



In this diagram, each extension (meaning everything to the left of the PBX) can connect to any other extension by dialing it directly. This means that if a modem were to send a fax to a local fax machine, it would be done by creating a direct connection between the devices through the PBX.

Line Trunking

Secondly, Asterisk offers line trunking. In its simplest form, line trunking simply shares access to multiple telephone lines. These telephone lines are usually used to connect to the global telephone network, known as the Public Switched Telephone Network, or PSTN, but can also be private lines to other phone systems.

These connections can be a single analog trunk, multiple analog trunks, or high-capacity digital connections that allow multiple concurrent calls to be carried on a single connection.

Telco Features

Asterisk supports all of the "standard" features we would expect from any telephone company (or telco). Asterisk supports sending and receiving Caller ID, and even allows us to route calls based on the Caller ID. Using Caller ID with the PSTN requires us to subscribe to that feature with our PSTN connection provider.

Asterisk also supports other features as expected, such as call waiting, call return (*69), distinctive ring, transferring calls, call forwarding, and so on. These basic features and more are provided by Asterisk.

Advanced Call Distribution

Asterisk can receive a phone call, look at attributes of the call, and make routing decisions based on that. If enough information is not supplied by our PSTN connection provider, then we can ask the caller to input the information, using a touch-tone phone.

Once we make a decision how to route a call, we can send them to a single extension, a group of extensions, a recording, a voicemail box, or even a group of telephone agents who can roam from phone to phone. We can use call queues to more effectively serve our customers while maintaining operational efficiency.

This flexibility gives us the ability to move from just having a phone system to creating powerful solutions that are accessed through the telephone. Advanced Call Distribution (ACD) empowers us to serve our customers in the best way possible.

One major differentiating factor between Asterisk and other PBX systems that support ACD is that Asterisk does not require the purchase of a special license to enable any of these features. The limit on how many call queues, for example, is determined only by the hardware we use.

Call Detail Records

Asterisk keeps complete Call Detail Records (CDR). We can store this information in a flat file, or preferably a database for efficient look up and storage. Using this information we can monitor the usage of the Asterisk system, looking for patterns or anomalies that may have an impact on business.

We can compare these records to the bill that the phone company sends out. They allow us to analyze call traffic, say to run a report to find the ten most commonly dialed phone numbers. We could also determine the exchange that calls us most frequently so that we can target our marketing to the right area.

Even more than that, we can look at how long phone calls are taking. We can count how many calls a specific agent answers and compare with the average. The uses of this feature are many.

Using this information, we can also identify abuses of our long-distance calling service. Employees all around the world steal long distance and time from employers; Asterisk gives us the tools to detect these possible causes of waste. The importance of calling records should not be underestimated: this information is invaluable for a variety of business functions. As many countries operate a national do-not-call list, we can quickly determine if we have called anyone on the list to ensure that our verification and checking processes are adequate.

Call Recording

Asterisk gives us the ability to record calls that are placed through the PBX. We can use this to provide training materials, as examples of calls that went badly or went well. This can also be used to prove call content to satisfy customers or partners as well as being potentially helpful in a legal situation. It's important to consider this feature when setting up your Asterisk service as you may have substantial hardware and storage issues to address if your PBX is destined to handle and record a substantial number of calls.

As a word of warning: Asterisk provides the feature. It is up to us to determine if it is legal, appropriate, and helpful to employ it in our particular circumstances.

Asterisk is an IVR System

Interactive Voice Response, or IVR, revolutionizes just about every business it touches. The power and flexibility of a programmable phone system gives us the ability to respond to our customers in meaningful ways.

We can use Asterisk to provide 24-hour service while reducing the workload for our employees at the same time. Asterisk allows us to play back files, read text, and even retrieve information from a database. This is the type of technology you come across in telephone banking or bill payment systems. When you call your bank you hear a variety of recordings and issue commands usually using a touch tone telephone. For example you may hear greetings and status messages, type in your account number and other personal information or authentication credentials. You will also often hear personalized information, which will be retrieved from a database, such as your last few transactions or your account balance. Systems such as this can be, and have been, implemented using Asterisk.

Asterisk is a Voicemail System

Asterisk has a fully-functional voicemail system included. The voicemail system is surprisingly powerful. It supports voicemail contexts so that multiple organizations can be hosted from the same server. It supports different time zones so that users can track when their phone calls come in. It even provides the option to notify the recipient of new messages via email: in fact, we can even attach the message audio!

Asterisk is a Voice over IP (VoIP) System

Asterisk gives us the ability to use Internet Protocol (IP) for phone calls, in tandem with more traditional telephone technologies.

Choosing to use Asterisk does not mean that we can only use Voice over IP for calls. In fact, many installations of Asterisk do not even use it at all. But each of those systems has the ability to add Voice over IP easily, at any time, with no additional cost.

Most companies have two networks: one for telephones, and one for computers. What if we could merge these two networks? What would the savings be? The biggest savings are realized by reducing the administrative burden for Information Technology staff. We can now have a few experts on computing and networking, and since telephony will run on top of a computer and over our IP network, the same core knowledge will empower our staff to handle the phone system.

We will also realize benefits from decreased equipment purchasing in the long run. Computer equipment gets progressively cheaper while proprietary phone systems seem to remain nearly constant in price. Therefore, we may expect the costs for network switches, routers, and other data network equipment to continue to decrease in price.

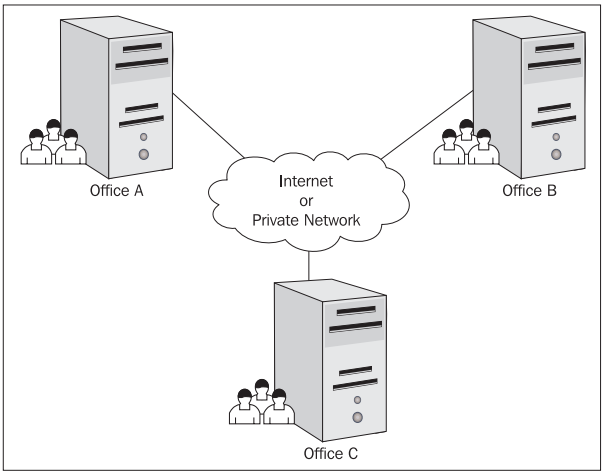
In most current phone systems, extensions can only be as far away as the maximum cabling length permitted by the telephone system manufacturer. While this seems perfectly reasonable, sometimes we would like it not to be so. When using VoIP we can have multiple users using the same Asterisk service from a variety of locations. We can have users in the local office using PSTN phones or IP phones, we can have remote VoIP users, we can even have entire Asterisk systems operated and run completely separately but with integrated routing.

One way to slash overhead is to reduce the amount of office space required. Many businesses use telecommuting for this purpose. This often creates a problem: which number do we use to reach a telecommuter? Imagine the flexibility if telecommuting employees could simply use the same extension when at home as when in the office or even when using their mobile!

Voice over IP allows us to have an extension anywhere we have a reasonably fast Internet connection. This means employees can have an extension on the phone system at home if they have a broadband connection. Therefore, they will have access to all of the services provided in the office, such as voicemail, long-distance calling, and dialing other employees by extension.

Just as we can bring employees into the PBX from their homes, we can do the same for remote offices. In this way, employees at multiple locations can have consistent features, accessed in exactly the same way, helping to ease the burden of training employees.

But this is not all that Voice over IP can give us. We can use an Asterisk server in each office and link them. This means that each office can have its own local lines, but office-to-office communications are tunneled over the Internet. The savings to be realized by avoiding call tolls can be significant. But there's more.



Once we have our offices linked in such a way, we can handle calls seamlessly, irrespective of which office the employees are in. For instance, if a customer calls Office A to ask about their account, and the accounting department is in Office B, we simply transfer the call to the appropriate person at the other office. We don't have to care about where that other office is. As long as they have a reliable internet connection, they don't even have to be in the same country.

We can route calls based on cost. If it is more cost effective, we can send our calls to another office, where the remote Asterisk server will then connect them with the regular phone network. This is commonly referred to as "Toll Bypass".

Another benefit of linking our phone systems together is that we can route calls based upon time. Imagine we have two offices, each in different time zones. Each office will probably be open at different times. To handle our customers effectively, we can transfer calls from a closed office to one that is open. Again, since we are using an Internet connection to link the offices, there is no additional expense involved in doing so.

By linking our offices together using Voice over IP, we can increase our customer service while decreasing our expenses: a true win-win situation.

The existence of all these options doesn't necessarily mean we should be using them, but with the versatility of Asterisk we may use and ignore options as it suits our requirements. If we were to use every single line type and feature that Asterisk supports this could lead to a very complicated and difficult-to-administrate system. We should choose the subset that fits our requirements and which would function well within our current communications setup.

What Asterisk Isn't

Now that we've discussed what Asterisk is, we need to discuss what Asterisk isn't. By seeing what Asterisk doesn't do, we can evaluate how important these pieces are to us, to help us determine if Asterisk is right for us.

Asterisk is Not an Off-the-Shelf Phone System

There are phone systems that can be ordered that are so easy to install, configure, and use that anybody without any training could do it. Asterisk is not one of them.

Asterisk's flexibility and robust feature set necessitate a host of configuration options. The best set of options to use will vary between installations, and sometimes vary within the same installation depending on the use. For instance, some handsets should have call waiting, while for other users, it is nothing but a distraction.

We can configure anything we need to with Asterisk, but there is a learning curve associated. In fact, sometimes there is programming involved in changing an attribute of the phone system. This is certainly something to consider.

While Asterisk in and of itself is not an out-of-the-box solution, there are packages based on Asterisk that are. For instance, a system called Asterisk@home is a single-CD installation that installs Linux, Asterisk, and a number of automated configuration tools. These tools allow the easy configuration of extensions, lines, and a few other features; however, to make this work, certain other features are not available.

Asterisk is also offered by companies that will customize the system specifically for your needs. These companies sell a server, the software, and the handsets at a package price, much as we see with proprietary phone systems. The difference is that the GNU Public License guarantees that we can view and modify the source code.

So, Asterisk in its purest form is not an off-the-shelf telephone system, although it is flexible enough to be used as one.

Asterisk is Not a SIP Proxy

Asterisk supports Session Initiation Protocol (SIP) for VoIP. Calls can be made and received with SIP using Asterisk.

In SIP, devices register with a SIP server. This server allows devices to locate each other to establish communications. When large numbers of SIP devices are used, a SIP Proxy is often employed to handle the registrations and connections in an efficient way.

Asterisk, however, cannot act as a SIP Proxy. SIP devices can register with Asterisk, but as the number of SIP devices increases, Asterisk is not able to scale very well. Therefore, if we intend to use over about 100 SIP devices, Asterisk may not be appropriate.

While Asterisk is not a SIP proxy, Asterisk can be configured to use one for registrations. One commonly used proxy is SIP Express Router, or SER. SER is an open-source SIP proxy that helps Asterisk scale in very large installations.

Asterisk Does Not Run on Windows

At one point, Asterisk had a demonstration CD that worked with Windows; however, Asterisk does not run on the Microsoft platform. Asterisk requires near real-time access to system resources. It also requires hooks into certain resources. As such, Asterisk is built to use Linux, the open-source *NIX operating system.

Is Asterisk a Good Fit for Me?

Looking at what Asterisk is and is not, the natural question follows: is Asterisk right for me? This is a vitally important question that should be given serious consideration. Let's take a moment and look at some of the considerations we must explore before we commit to using Asterisk.

Trade-Offs

There are a series of trade-offs we must consider with Asterisk. Choosing Asterisk will lock us into certain choices, while others will be available whether we install an Asterisk server or not. We will now examine some of these trade-offs so that we can gauge the impact they have on us.

Flexibility versus Ease of Use

Asterisk is a very powerful framework into which we can install almost anything. We can configure each piece of Asterisk to the minutest detail. This gives us an amazing amount of flexibility.

This flexibility does not come without a price. Each of these details must be researched, understood, and tried. Each change we make affects other parts of the phone system, whether for good or for bad. Asterisk is not an easy-to-use platform, especially for the beginner.

There is a learning curve, but it is one that can be surmounted. Many developers have become experts in telephony; many telephony experts have mastered server administration. But each of us must decide what we expect from our phone system. I like to think of it in three major categories, as outlined in the following table:

Description	Solution
I want to plug the telephone system in and never think about it again. I want someone to call when things are not working. I do not plan to add anything to the system once it is set up.	A proprietary phone system is probably your best bet. Many offer a pre-configured system, and when changes are made, a certified consultant will be required.
I don't know much about phone systems, but I want to learn. I need a phone system soon. I'd like to have flexibility and additional features, and may change the configuration of my phone system from time to time.	Either use a packaged version of Asterisk or have a consultant build a customized Asterisk server. Learn to use Asterisk. Build a couple of Asterisk servers just to explore. Add features as necessary.
I want to learn and build my own phone system. I am interested in creating a custom solution for my problems. I am willing to accept the responsibility if something doesn't work, and take the time to figure out why.	Build an Asterisk server from the ground up. Much will be learned in the process, and the result will be an extremely powerful business tool.

These are, of course, not distinct categories. We each fall into a continuum. It is important to realize that Asterisk, as great as it is, is not the right solution for everybody. Like any technology we implement, we must consider its impact on the business, and decide whether it will become something useful that enables us to work better, or whether it will require too much maintenance and other work to make it an efficient addition. This depends entirely on our purposes and the other technology we have that requires our attention.

Graphical versus Configuration File Management

Asterisk currently uses plain text files to configure most options. This is a very simple way to create, back up, and modify configurations for those who are comfortable with command-line tools.

Some PBX systems offer a Graphical User Interface (GUI) to update the configurations. Others don't allow the configuration to be changed except by dialing cryptic codes on telephone handsets. Still others cannot be configured at all, except by certified technicians who receive the required software and cables from the phone system manufacturer.

A few good open-source tools are being created to ease the management of Asterisk. However, to get the full ability to customize Asterisk, editing of text files is still required. To help get used to this method of configuration, this book focuses on the text files and does not rely on any GUI package.

As we evaluate Asterisk, we must ask ourselves if we are happy about working with text configuration files to configure our phone system. If we are unwilling or unable to do so, Asterisk may not be the best choice.

Calculating Total Cost of Ownership

Asterisk is distributed as free, open-source software. The only costs involved with Asterisk are hardware, right? Well, maybe not.

As we have been discussing, Asterisk is very flexible. Determining how best to use the flexibility can quickly become a huge time-sink. Compatible handsets are also not free. If we are going to use the G.729 protocol, which compresses VoIP traffic by a factor of eight while maintaining excellent voice quality, we will also have to pay licensing fees.

With commercial phone systems, the costs are typically higher than with Asterisk; however, they are a fixed, known constant. Depending on the way we use Asterisk, costs can vary greatly.

The total cost of owning Asterisk can also include downtime. If we choose to support Asterisk on our own, and have to work to try to get Asterisk back up after a failure, there is an opportunity cost involved in the calls we should have received. This is why we should only choose to support our phone system internally if we have the appropriate resources to back that up.

Total Cost of Ownership (TCO) is not an easy calculation to make. It involves assumptions of how many times it will break, how long it will take us to get it back up and running, and how much consultants will charge us if we need their services.

TCO is only useful when comparing phone systems to each other. The following elements should be included when comparing TCO of multiple phone systems:

- **Procurement Cost**: This is the cost to buy the PBX. In the case of Asterisk, it is only the cost of the hardware; other systems will include an element of licensing.
- **Installation Cost**: This is the cost to configure and deploy the PBX. Some companies choose to do the deployment in-house; in such instances, there is still a cost, and to enable fair comparisons, it should be included.
- **Licensing Cost** (one-time): This is the cost of any one-time licensing fees. Some PBX systems will require a license to perform administration, maintenance, connect to a Primary Rate ISDN line (PRI), etc. In Asterisk, this would include the G.729 licensing cost, if required.
- **Annual Support Cost**: This is the estimated cost of ongoing maintenance. Of course some assumptions will have to be made. To keep the comparison fair, the same assumptions should be carried over between vendors.
- **Annual Licensing Cost**: Some phone systems will have an annual cost to license the software on the handsets, as well as a license to be able to connect those handsets to the PBX.

When we have created the table, we can calculate the TCO for one year, two years, and so on. We can then evaluate our business and decide what costs we're willing to incur for our phone system.

Return on Investment

The cost of owning a phone system is only one piece of the Return on Investment (ROI) puzzle. ROI attempts to quantify an expenditure's effect on the bottom line, usually used to justify a large capital outlay.

Just as an example, one phone system that I installed went into an existing business. Its existing phone system had an automated attendant that had the unfortunate habit of hanging up on customers if they pressed the 0 key, or if they didn't press any key for 5 seconds.

What was the ROI for moving to a new phone system? Not having angry customers who got hung up on is a hard value to calculate. According to one of the owners of the business, that value was infinite. That made the cost of Asterisk very easy to justify!

ROI is basically the TCO subtracted from the quantification of the benefit (in money) to the business. So, if we calculated that a new phone system would save $5000 and cost $4000, then the ROI would be $1000.

Another interesting calculation to make, which is also categorized as ROI, is the time for the cost to be recouped. This calculation is one that I find helpful in making a business case for Asterisk.

Suppose a phone system costs $5000 to install. Using toll bypass, you can save a net $500 per month. In 10 months, the cost of installing the system will be swallowed up in the savings.

These are simple examples, but Return on Investment can help to justify replacing an existing phone system. By having these numbers prepared before proposing to replace the phone system, we can have a more professional appearance and be more likely to succeed in starting our Asterisk project.

Summary

Asterisk is a powerful and flexible framework, based on open-source software. It can be used to create a customized PBX for almost any environment. But it is not always the best choice for reasons we have just explored. We must consider this carefully in order to be confident that Asterisk is the right choice for our situation and ensure that the time and money invested in setting up the Asterisk service is a worthwhile outlay.

PACKT
PUBLISHING

2

Making a Plan for Deployment

Now that we have chosen Asterisk to meet our needs, we need to determine our course of action. We will go through some common requirements, discuss the most common choices for solutions, and finally make an informed decision. As we go along, we should make notes to help us on our way.

The Public Switched Telephony Network (PSTN)

Most of the telephones in the world are connected to a vast network, enabling any telephone to reach any other. This network is called the Public Switched Telephony Network (PSTN). The phones that are on it are all reachable by dialing a number, which may include country codes, area codes, and telephone numbers.

While there are instances in which interconnection with the PSTN is inappropriate, most users of telephones have the expectation that they can reach the world at large. Therefore, we will consider interconnection to the PSTN as a requirement.

Connection Methods

There are a number of different methods to connect to the PSTN. Each has advantages and disadvantages, most of which we will touch on. Since pricing varies depending on city or country, exact pricing will not be given. Pricing should be researched based upon the location of the Asterisk server.

We will handle each connection method one at a time.

Plain Old Telephone Service (POTS) Line

Probably the most common connection to the PSTN is a POTS line. This is an analog line, provided by a telephone carrier. Each POTS line can carry only one conversation at a time.

For small installations, POTS lines are usually the most cost-effective when connecting directly to our Local Exchange Carrier (LEC), a term used to refer to any company providing local telephone service. Eight lines is usually the point at which we should look seriously at another technology for our connection.

POTS lines from our LEC require a Foreign eXchange Office (FXO) interface to be usable in Asterisk. We will focus on Digium's offerings, namely the FXO module on a TDM400P. Each TDM400P can use up to 4 modules; therefore, if we have 1 line, we will have 3 empty module slots on the card.

Integrated Services Digital Network (ISDN)

ISDN is an all-digital network that has been available for over a decade. It is available in two major versions: Basic Rate Interface (BRI) and Primary Rate Interface (PRI).

ISDN divides a line into multiple channels. Each channel can contain either payload (Bearing, or B channel) or signaling (Data, or D channel). A BRI has 3 channels: 1 D channel and 2 B channels. Therefore, two phone calls can be in progress at a time on a single BRI. A PRI has 24 channels: 1 D channel and 23 B channels, resulting in up to 23 simultaneous calls.

ISDN is not limited to voice alone. Each channel can carry 64k of data, if so configured with the LEC. This gives ISDN a lot of flexibility over POTS lines, as the channels can be reconfigured between voice and data on the fly.

With its separate D channel, ISDN is able to do things POTS cannot, such as setting custom caller ID, receiving dialed number information, on-the-fly redirection of calls, and a whole host of other cool features. Of course, all of these features require cooperation from the LEC, which is not always forthcoming.

BRI does not have high penetration in the United States market. Some accuse LECs of vicious pricing, while others claim consumers are to blame for fearing new technology. Either way, the result is the same: if we call our LEC and request a BRI, they will assume it is for data.

PRI, on the other hand, is widely used in the US. It is the connection of choice for larger installations. PRIs are actually delivered over T1 connections, a proven technology. Although the author has many contrasting stories, T1s are usually very reliable.

T1 or E1

Technically speaking, when ordering service from an LEC, we order a DS1, which is delivered over a line referred to as a T1. However, this detail is usually overlooked. Therefore, we will refer to it in its vernacular: a T1.

A T1 is a line with 24 channels. Each channel can contain a call. Therefore, a T1 can contain an additional call when compared with a PRI. In Europe, E1s are more common. Much like a T1, they have 32 channels instead of 24. T1s have to signal the call somehow. They way they do this is through Robbed Bit Signaling. What this means is that a bit is robbed from time to time, as information needs to flow about the connection. While this is usually imperceptible to the human ear, it can be deleterious to data connections.

Using a T1 to deliver both data and voice is common. Some of the 24 channels are designated to be used for data, and others are used for voice. There may even be unused channels. LECs are able to offer lower pricing when bundling services in this way, as a few channels may be voice, others used for an Internet connection, and yet others could be used for a private data connection to another office.

LECs are able to send information about the number that was dialed at the beginning of the call. In this way, one advantage of the PRI has been matched by T1s. If we intend to have about 8 to 12 lines as well as a Data connection, a T1 can be a good choice.

Voice over IP Connections

In recent years, a new way to connect to the PSTN has cropped up. Companies are using PRIs, T1, and other technologies to connect to the PSTN, and then resell those connections to consumers. The users connect to the companies offering these connections through Voice over IP technologies. By so doing, we can skip dealing with LECs completely.

This service is called Origination and Termination. Through these services, we can receive a real telephone number, with the area code depending on what the provider has access to. Not all providers can offer numbers in every locality. This means that our number could be long distance from our next-door neighbor, yet local to someone in the next state. The advantage of this, however, is that the provider will route most of the calls over their VoIP infrastructure and will then use the PSTN when they get to their most local point at the receiving end, which can mean that long distance charges are dramatically reduced. If we call a variety of countries, states or cities it can be worthwhile to research a provider that offers local PSTN access to the areas we call most.

The rates per minute are usually very attractive. Often, long distance is at the same rate as local calls. One thing to watch out for is that some providers charge for incoming minutes, much like on a cellular telephone, and some providers also charge for local calls.

Another thing to be aware of is that some providers require that you use their Analog Terminal Adapter (ATA). This means that they will send you a box that you plug into the Internet, which speaks Voice over IP. Then, you have a POTS line to connect a phone (or Asterisk) to.

Voice over IP makes sense in many installations. But for the quality to be acceptable, a reliable Internet connection with low latency is required. Another thing to watch out for is jitter. Jitter refers to the variation in latency from packet to packet. Most protocols can handle latency a lot better if it is constant throughout the call.

A good candidate for Voice over IP is a site where interruptions in service will not endanger life and will not irreparably harm the company. While VoIP providers strive to achieve very high availability, we also have to rely on the Internet at large and our VoIP provider's ISP, as well as our own ISP.

If our telecommunication needs are such that periodic downtime is tolerable, VoIP will probably be our least expensive option. It requires less hardware in our Asterisk system as well, increasing the savings: to use VoIP with Asterisk all we need is a system capable of Internet access; we don't require any specialized telephony hardware.

Determining Our Needs

Now that we have examined some of the options, we need to determine what our needs are. Requirements will vary quite a bit from site to site. Something to keep in mind is that, although the previous choices are distinct, they can be mixed in an Asterisk installation. We can have VoIP providers and POTS lines, as well as a PRI if we desire. It's very common to have this type of setup; for example if we have an office in another country we can call them using VoIP but all local calls could use POTS. It is important to understand the calls our system will be making and where they will be going, so that we can arrange for the necessary services and ensure that the calls are routed accordingly. If we have an existing telephony system, we can take a look at the calls it's making just now and our current costs so that we can determine what technologies will be of most use to our system's users.

Now is the time to begin documenting what our plan is. If Asterisk is to replace an existing system, then we should start by writing down all of the current lines coming into our incumbent PBX. Once that is done, we need to look at what our requirements are.

First, we need to determine how many lines are needed. Telephone providers can generate a usage report that will tell us the maximum concurrent connections we have experienced in the last month. While they are able to do this, many providers are not very happy to have to run such a report; however, without that information we have nothing to gauge our needs other than gut feelings.

If we need more channels than we have, someone will get a busy/congested signal. Therefore, we should plan to have the maximum number of channels we have used plus a reasonable cushion. 125% of our current maximum is usually a reasonable cushion, this allows for instant 20-25% growth so that we can accommodate a sudden increase in calls without the system failing over, causing busy signals. If we do increase calling to this level for a relatively long period, we must consider an increase in lines to prevent congestion. These numbers are a guideline and they can change depending on circumstances. In a call center where the main business purpose is to make and receive calls, 150% may be a more satisfactory figure. We also should take into account the time it takes to get new lines set up from our local operator. If a significant event occurs that generates a large number of calls we should have the capacity to handle this or be able to increase capacity quickly.

Now that we have a number of lines, we need to determine the technology to use for each. VoIP is usually cheapest, especially for long-distance calls. PRI is usually the most reliable, and for incoming calls is often cheaper than VoIP.

While pricing the options, we need to remember that POTS lines usually only have a single phone number, while a PRI can have hundreds of phone numbers. If we are a business that receives only a few calls, but needs the calls to have different phone numbers, then a PRI probably makes the most sense. Also with a PRI we can trunk more effectively, which may become essential.

Although a PRI can have hundreds of phone numbers, there is a charge for each number each month. Called DID (Directed Inward Dialing) numbers, these "virtual" numbers are usually sold in blocks of 10-20. If we do not order enough to begin with, it is usually not difficult to get new DIDs ordered; often, they can be available the same week, depending on the phone company. We assign these numbers to individual devices or groups of devices ourselves, once we have them allocated. This means we can decommission or reallocate numbers as necessary. We may have campaign DIDs that are reassigned to different groups depending on the current campaign, personal DIDs for key staff or our main DID, which would probably be assigned to a group of people responsible for handling these calls.

We should take this opportunity to write down what lines we want, what phone numbers we need, and get quotes if it differs from the currently installed PSTN connections.

Terminal Equipment

Now that we have decided on our PSTN interconnection, we need to decide on our internal connections. Our PBX can have modems, fax machines, hardware and software telephones, and other PBXs connected. We will refer to all these different machines as Terminal Equipment.

Types of Terminal Devices

There are four major types of Terminal Equipment: hard phones, soft phones, communications devices, and PBXs. We will cover each type briefly.

Hard Phones

The term Hard is short for Hardware. Hardware phones are physical devices that act as a telephone handset. Hard phones are available for POTS (as used in the typical household) or VoIP.

Voice over IP uses various protocols, depending on the handset, PBX, and the requirements. The major protocols supported by Asterisk are as follows:

H.323

The first protocol we will be looking at is H.323. Formally known as "ITU-T Recommendation H.323: Packet-based multimedia communications systems", this is a suggestion on how to accomplish conferencing over IP, which includes voice, video, and data. This recommendation actually came out at about the same time as SIP but has been more widely implemented.

The H.323 standard enjoys full backwards compatibility. Currently H.323v5 is out, and v6 is being discussed. Each new release keeps all of the pieces of the previous version. This gives a clear upgrade path and some assurance that equipment won't be quickly antiquated.

H.323 equipment is widely available. From gateways to telephone handsets, all of the needed equipment is relatively easy to find. Most of the telephone handsets are full-featured because the H.323 protocol has a robust feature set.

While the H.323 standard was not designed for wide-area networks, a whole set of rules allowing cross-domain addressing have been created. A system for reporting Quality of Service (QoS) back to a server has also been developed, allowing such information to be used to route future calls.

Finally, H.323 as a standard supports call intrusion. New endpoints can be added dynamically to any conference (i.e. a call) at any time.

Asterisk support for H.323 is not built in. Instead, an additional package, `asterisk-oh323`, must be installed. After installation, H.323 handsets and gateways can be addressed much like any other channel in Asterisk.

SIP

The Session Initiation Protocol, or SIP, is another method of signaling VoIP calls. SIP is part of the default installation of Asterisk.

Most of the newer VoIP equipment is supporting the SIP protocol. It has a number of advantages. One such advantage is that the code is smaller. The reason for this is that SIP only supports very basic features. All advanced features are supported through separate Internet standards. Another reason for its small footprint is that, as features are deprecated, the code to implement them is ousted.

Another advantage of SIP's design is its modular nature; as such, extending the protocol is easier to do. It also scales better and was designed with a large network in mind.

SIP seems to be the future of VoIP. There are many features that H.323 has but which are not available on SIP, though. This includes handset conference control, better Media Gateway definitions, and data sharing. However, SIP is a very good protocol for simple phone calls. Also, since we are using Asterisk, conferences are controlled by Asterisk, not the handsets. Asterisk is a clear Media Gateway, and when used as such, the ambiguity in SIP is not an issue.

IAX

The Inter-Asterisk eXchange (IAX) protocol is a protocol created by the programmers who brought us Asterisk. Because of the limitations of SIP and H.323, they chose to create a new de facto standard that would allow Asterisk servers to accomplish many things that are simply impossible with the other standards. They also support some features that are extremely difficult to do in SIP and H.323.

First, IAX pierces Network Address Translation (NAT) easily. Most firewalls and home Internet gateways use NAT, as well as some service providers. SIP and H.323 have worked hard to develop standards to allow them to break through the different types of NAT; however, IAX can work through most NAT devices right out of the box.

IAX is more configurable than the other protocols when dealing with Asterisk. Since the source code is available, we can modify it if we so desire, and then submit those changes to be evaluated for inclusion in future versions of Asterisk. Since IAX is not currently an Internet standard, per se, there is no standards body to work through, allowing more rapid improvement and growth.

IAX supports the trunking of calls. This means that multiple calls can be combined through a single stream. Through the trunking capability, a significant amount of bandwidth can be saved by not having the overhead of multiple streams.

IAX connections between servers support the switch command, with which information on how a call is routed can be efficiently shared between Asterisk servers.

IAX supports a large number of codecs. Any codec supported in Asterisk can be used with channels of this type.

Because IAX is an Asterisk-created protocol, there are not many handsets and gateways available. However, as time goes on, more and more devices are supporting the IAX protocol.

Just as a note, we sometimes see IAX and IAX2 differentiated. IAX2 has been merged into IAX, and IAX has been deprecated. Thus, if a device claims to support IAX2, it should really be supporting IAX.

Soft Phones

Much as hard phones are phones implemented in hardware, soft phones are phones implemented in software. Using all the same protocols available to hard phones, soft phones are far less expensive to implement. By using the general-purpose computing resources of a personal computer, the expensive proposition of replacing all telephones in a building can be avoided.

Before going further, we should recognize that most hard phones are in all actuality soft phones combined with bit of special-purpose hardware. The computing power of a hard phone is not as vast as that of a PC, but unlike a PC is specially tuned for carrying voice. Thus, we should not dismiss use of hard phones immediately.

The sound quality experienced on a soft phone will depend greatly on the available resources on the PC, the quality of the software used, and the quality of the data network between the PC and our Asterisk server.

Soft phones will have a hard time being accepted by some users, it is true. In addition to the political issue of having people use their computer to talk on the phone, we also have to address disaster planning. If we lose power, keeping a computer up that draws in excess of 400 watts will be far more difficult and costly than keeping power to a hard phone that draws 15 watts, especially for prolonged outages.

The most significant advantage of the soft phone is cost. In most businesses, desks contain a computer and phone at a minimum. If you can remove the phone there is an obvious reduction in hardware costs. There are a variety of soft phone products available and most operating systems come with a basic soft phone package by default. There are also a variety of open source products available. The choice of product, soft or hard, is equally as important as the PBX. You must be sure that the users will use the device and be sure that it will be reliable and supportable.

Communications Devices

Dedicated communications devices, such as modems and fax machines, are still very prevalent in business today. While these devices could be replaced with more modern, more reliable, and faster technologies, the new technologies have not yet been embraced.

Most of these devices will be analog (meaning they will require a POTS line). As mentioned before, a T1 can connect 24 lines, and a POTS line can only connect 1 line. With a device called a channel bank, a T1 can be split out into 24 POTS lines. When we require many POTS lines, channel banks are usually cost effective.

Communications devices do not all use analog signaling. One such device would be a T.38 fax gateway. This protocol allows regular faxes to be sent over UDP. At this time, Asterisk does not support the T.38 protocol, but hopefully will soon.

One extra note about faxing: Asterisk supports receiving and sending faxes via an add-on called SpanDSP. With this, Asterisk can receive a fax and turn it into a TIFF file. This TIFF file can then be further processed to become a PostScript or PDF file and emailed to the proper recipient. The installation of this add-on is not covered here, as it is changing rapidly.

These communications devices are usually supported for legacy reasons. We should continually strive to reduce outdated technologies and replace them with up-to-date solutions.

Another PBX

We can connect PBXs together to provide services to users hosted on another PBX. We can use SIP, PRI, T1, H.323, or IAX to connect the PBXs.

If we are connecting multiple Asterisk PBXs, we should use IAX. The IAX protocol has a number of features with this specific use in mind, such as the ability to have multiple conversations trunked into the same UDP stream, yielding greater efficiency.

Choosing a Device

Now that we have seen the broad offerings of terminal devices, we see how difficult it can be to choose one to meet our needs. After choosing a type of device, we then have to choose a manufacturer and model. This task can be daunting. Let's take a few minutes and discuss how we will make the best decision based on the available information.

Features, Features, and More Features…

As we review available phone handsets, we will be inundated with all the features that manufacturers can throw at us. These lists are overwhelming, even to the most seasoned experts. It is very difficult to compare two handsets solely on features, as some features have different names.

Determining the usability of a particular phone handset should be a straightforward process. This process has four major steps: requirement elicitation, prioritization, and documentation, followed by handset testing.

Requirement Elicitation

This is the brainstorming step. We should go to each user and determine what his or her needs are. We ask the user what features he or she uses on the current phone. We observe that person working for a period of time to get a good sampling of what he or she actually does.

We should then go to the user's manager and see what a person in that position is expected to do. We add these features to our list. While this list will be unique to each user, many will be very similar. We should see patterns of usage emerge between groups of employees.

Requirement Prioritization

In this step, we take our requirements list from the previous step and, working with the user and manger, determine which features are used most, which are most important to that user's role in the organization, and which features are simply nice to have. We should also attempt to recognize any deficiencies in the current technology. Changes are often embraced if the change adds value to the user by making a task easier or in some cases removing a task entirely. It's important that we recognize all nuances of the current system in order to provide the user with a replacement that will suit them.

We then should create a quantitative scale for each feature. For example, if we were working with an operator, a transfer button would be a 10, while a Do Not Disturb button will probably be a 1. If we had a phone handset with both, it would score an 11. By putting numbers on the required features, we can come up with a quantitative answer to a very subjective issue.

Requirement Documentation

This step is most important of all of the steps thus far, especially for consultants. We take the list of requirements and their weights and write them in a short document. We then have the user and the manager sign it off to indicate their agreement.

This may seem a little formal for picking a telephone handset, but it is an effective method of communicating expectations and plans between you, the implementer, and the users. This can help to prevent surprises or differing recollections of what was promised.

Phone Testing

This is the final step. After comparing the available handsets against the document we created in the previous step, we choose the highest scoring handset. We then take a handset of that type to the user and have him or her use it (if we have a test system installed by this point) or at least sign off on it conceptually.

Again, this is an opportunity to ensure our users' expectations are reasonable, that commitments are clearly defined, and that our users are kept informed during the decision-making process. It can also help us get buy-in from the users as we make the major adjustments that will invariably accompany a new phone system.

Determining True Cost

When we look at what handsets to compare our requirements document against, the issue of cost will have to be looked at. Before we offer a handset that would not be possible under our project budget, we should determine that the handset meets all of the requirements of the business, which includes the element of cost.

The issue of cost is not as simple as looking at the retail price of a handset. Each type of phone will have multiple types of cost. These costs will usually fall into one of the following categories:

- **Handset cost**: This is the easiest cost to determine. It is the actual amount of money that will have to be spent to acquire the telephone.
- **Port cost**: This cost is the element of what the phone connects to on the other end. On a VoIP phone, for instance, this could be a portion of the cost of a new network switch that supports Quality of Service (QoS) to enable reliable voice communications.
- **Headset cost**: If a phone will require a headset, then we should consider the cost of that headset as we choose the phone. Different connectors are available depending on the model.

- **Software license cost**: Some phones will require the purchase of G.729 licenses. Other phones may require a license for the software on the phone (usually referred to as firmware). We should not fail to consider this cost while computing the cost of the phone.
- **Installation cost**: Different phones require different amounts of time to install. That time translates into cost.

By considering each of these factors for each different handset, we get an idea for the true cost of each particular phone. With all of these costs defined, we can see which phones are within our budget and which are simply too expensive.

Compatibility with Asterisk

Not all handsets interoperate equally with Asterisk. Referring to the Asterisk Users mailing list archive, we can ensure that no serious incompatibilities have been discovered. Also, a wiki is available at `http://www.voip-info.org`. A vast array of useful information about Asterisk is available there. The site is searchable and is constantly updated.

We do not have to select a single protocol for all VoIP phones. Instead, we can mix and match protocols to our best advantage, thanks to the flexibility and power of Asterisk.

Sound Quality Analysis

Sound quality is a very subjective thing. Each user will have a personal threshold between acceptable and unacceptable.

Each phone will have varying sound quality. The variables that can affect the quality of a call are staggering. Network latency can significantly affect sound quality, but so can configurations of the phone. Determining what the cause of low sound quality is can be difficult to do.

Build quality from a manufacturer can also affect quality. When wide variations are allowed from one phone to the next, the result is usually inconsistency from handset to handset. Thus, we have to choose a manufacturer we can trust.

While there is no absolute, the quality of sound on telephone handsets, from highest to lowest, is usually as follows: analog hard phone, VoIP hard phone, analog soft phone, VoIP soft phone. If you are doing a comparison between different handsets, the main things to pay attention to are the amount of background noise (or hiss), distortion, drop outs, popping, and highly digitized voice. If we have users who are extremely sensitive to sound quality, analog will probably be our best bet. For those users who are a little more forgiving, VoIP allows us to use one network for our phones and our computers.

When determining what terminal equipment to use, we need to consider the sound quality of each device and match it against the needs and expectations of our users, and temper that with the cost of that device as compared to the budget.

Usability Issues

The world's most advanced VoIP handset is absolutely useless if our users cannot figure out how to use it. As we decide what equipment to provide for our users, we should consider where they are at in the continuum of technological awareness. While VoIP hard phones with context-sensitive buttons are useful for most users, some people find the interface confusing and frustrating.

This is one big issue that we need to address in the handset testing that we do after eliciting the requirements that our user has for a new phone. We have a duty to ensure that our users can use the handsets we choose. We must be careful not to assume that they will figure it out, as doing so often causes hurt feeling and resistance to change. The success of Asterisk will be largely measured by the response of our users.

Recording Decisions

It is time to decide what kinds of terminal equipment we will use with Asterisk. First, we should make a list of all users of our phone system. Based on the requirements we get from them and their supervisors, we should decide what type of device to use, whether it is a hard phone or a soft phone. Next, we should determine a protocol to use. Finally, determine a brand and a model of phone to use.

We should take the time to write this down. This list should be provided to the decision makers, and kept up to date as changes occur, which they inevitably will. Again, doing so will keep everybody informed and reign in the expectations to keep them reasonable.

How Much Hardware do I Need?

This is probably one of the questions most frequently asked by those who are new to the world of Asterisk. The answer depends largely on what we are going to do with our system.

Conversations that bridge between codecs (called transcoding) take the most power to handle. Voice over IP conversations seem to take a little more processing power than straight Time-Division Multiple-Access (TDM) calls. Having our server run scripts to find information will take more power than if we define everything statically. How many different conversations we have going at a time will affect how much horsepower we need our server to have. As will the features we use.

Do you see the complexity of answering this question? We have to figure out what we are going to use before we can figure out how big a server we will need. That said, there are some good rules of thumb we can start off with.

First, while we can run an Asterisk server on an old Pentium 90 with 64 MB of RAM, why would we want to? We are creating a robust phone system. We do not have to pay to license the use of the software, and we do not have to pay per extension. We can go spend some of the money we saved and buy a decently powerful server.

As we select the components for our server, we need to remember that we are not building an email server or a web server. We are creating a PBX that people are going to expect to be running *all the time*. We should select a stable chipset, with an up-to-date BIOS, and match it with other current high quality components. By using high-quality components, we increase the likelihood of ending up with a highly-availability phone system.

On another note, we should select a server with as much redundancy as possible. A RAID-1 controller could save our phone system in the event of a hard drive failure. A pair of RAID-1 controllers that are mirrored could save our phone system in the event of a controller failure or a PCI slot failure. A server with redundant power supplies will help us in the event of power failure or a power supply failure. Of course, our phone system should be on an Uninterrupted Power Supply (UPS). This is not only for protection from power failures; it will also protect from spikes, and often even lightning.

Depending on the reliability requirements, we might need a redundant server. There are hardware devices that will detect if a PRI is down and automatically failover. Then again, for most installations, this is overkill.

The most important lesson to keep in mind is that people have grown to depend on phone systems. We should not skimp on hardware as doing so could, in the long run, cost us dearly. With the unique pricing structure of Asterisk, all we will have to pay for is any additional hardware to get increased reliability and capacity.

Along with hardware, the question is often asked "Which distribution of Linux should I use?" If you already have experience with some distribution of Linux, you should be able to make Asterisk work with that distribution. Asterisk is very flexible and has been built with commonly available dependencies, and any distribution of Linux should work. That said, some distributions will require more effort to enable some features such as automatically starting Asterisk when the server boots. Since each distribution treats startup scripts differently, most distributions will require a minor amount of tweaking.

Also check the wiki at `http://www.voip-info.org` for more information on the distribution you intend to use. It has up-to-date notes on compatibility problems, caveats, known issues, and often workarounds for those issues.

Choosing the Extension Length

While creating our phone system, we will need to create a set of extensions. Although Asterisk has no such requirement, these extensions should probably all have the same length to give comfort to our users. We must determine the length that we will use for all of our extensions.

When creating extensions, it is often advantageous to group certain extensions together. For example, all sales extensions could be in the 200's, support in the 300's, management in the 100's, etc. Or we could go further and say that all first-tier support will be in the 3100's, the second-tier support will be in 3200's, third-tier support will be 3300's, and so on.

We should keep in mind that it is easier to add extensions when there is an available number than it is to renumber all extensions in a building because we have filled up all of our available dial strings. For instance, suppose we chose 1-digit extensions and have the following phone list:

```
0 – Operator
1 – Reception Desk
2 – Break room
3 – Conference room
4 – John
5 – Sally
6 – Jennifer
7 – Fax Machine
8 – Voicemail Access
9 – Outgoing calls
```

This system will work fine until we add another extension. When we add another extension, we will have to give a new extension numbers to all of our users.

Now consider the following phone list:

```
1000000 – Kitchen
2000000 – Bedroom
3000000 – Office
8000000 – Voicemail
```

In this house, for someone in the kitchen to call the office (think "Dinner's ready; will you please leave that computer and *come eat*?!?"), the user has to dial 7 digits to accomplish what could have been done with one.

Therefore, we need to be smart about how long we make our extensions. Often, if we are replacing a phone system, we should just adopt the numbering already in place to make the transition a little easier for our users. Some phone systems may not have had extension numbers before, such as old analog systems. All lines were simply visible from all stations. In those instances, we should be sensitive to the new learning that will have to take place and make the extension number length as small as possible.

We also need to consider some special instances. First, most people do not want an extension that begins with a 0. Simply put, nobody likes to be a nothing and having a leading 0 for anybody but the operator makes them feel emotionally put-down. Also, we should reserve all extensions beginning with a 9 as outgoing telephone calls. Add to that the need to provide services like call recording, conferencing, and voicemail access. We will give all such services a prefix, such as 8. Thus we see that we have already lost 30% of all of the available extensions.

A good rule of thumb in computing is to take what we believe we will use and triple it, and then round up. Thus, if we believe that at the height of our system, we will have 100 users, we should assume that we will have 300 users. If we believe we will never have more than 10 users, we should assume 30.

With this in mind, here is a table of what we will need:

Expected number of extensions	Our assumed number of extensions	Length to use for extensions
2	7	1
22	70	2
222	700	3
2222	7000	4

Keep in mind when reading this chart that it is much easier to have people dial an extra digit than it is to make them learn all new extension numbers. Thus, if we are a border case, we should go ahead and move on up to the next extension length.

Another idea that we can take advantage of is using an extension that gives a lot of information about the destination. Take for instance a corporation with 7 locations. The first digit in the extension could designate the location. Then the second digit can designate the department, and the remaining digit(s) can designate which member of the group is sought. Thus, knowing the structure and an extension can give an idea of where that person is and what he or she does.

In some environments, such information is not desirable. For instance, on a college campus, some employees work very late at night. If the extension gives their precise location, stalking and threats of physical harm can prove problematic. Therefore, we need to be sensitive to such concerns.

One alternative to these layouts is to use the last few digits of a phone number to refer to each extension. This can work very well if all of such digit strings are unique; however, it can cause problems. Suppose we chose a 4-digit extension and have the phone numbers 555-1234 and 777-1234. Which one is extension 1234? Or suppose we use 7 digit extensions and have (800) 555-1234 and (866) 555-1234. Which one is extension 5551234? Thus, some organizations have moved to a full 10-digit extension length. While this allows 10^{10} extensions, it can cause some users to complain about usability and convenience.

With the flexibility of Asterisk, we can choose many different ways to allocate extensions, all of which will influence our decision on extension length. We must balance our users' expectations with our desire to leave room to grow. By so doing, we can create extensions that are easy to maintain and user-friendly.

Summary

Now that we have decided to use Asterisk, we must make a plan. This chapter has looked at the different types of hardware that an Asterisk system needs, namely:

- What technology we use to connect to the PSTN
- What technology or technologies we use to connect our handsets to Asterisk
- What server hardware we will use
- How we will architect our extensions to be easy to use while also allowing for the growth we can realistically expect

As we draw up our plan, we must address each of these options, before moving on to the next stage, the installation of the Asterisk software itself, which we cover in the next chapter.

[PACKT PUBLISHING]

3
Installing Asterisk

We're making great time! Together, we have selected Asterisk to meet our needs, created a plan to define how our phone system will act, and we are ready to begin installing Asterisk.

Preparing to Install Asterisk

In order to install Asterisk, we will need a computer with Linux installed. It's a good idea to ensure your system is up to date, for instance using the APT tool. Once we have installed our distribution of choice, we need to make sure we have a few additional packages installed. The required extra packages over a base installation are:

- `bison`
- `gcc`
- `kernel-source`
- `libtermcap-devel`
- `ncurses-devel`
- `openssl096b`
- `openssl-devel`

For versions before 1.2, we must also install mpg123. There is a current security issue with mpg123. If an attacker were to correctly craft an MP2 or MP3 file, and then trick someone into playing back that file, arbitrary code could be executed. While it is preferable not to use packages with known vulnerabilities, the exposure here is minimal, as we should be the only ones creating music on hold for our system. To work around this issue, developers have implemented a native MP3 playing facility within Asterisk starting with version 1.2.

To install mpg123, enter the command `wget http://www.mpg123.de/mpg123/mpg123-0.59r.tar.gz` to download the tarball. Once you've got it, unpack it and navigate to the `mpg123-0.59r` directory, and then type `make linux` to compile the program, and `make install` to install it.

Once we have installed these packages, we are ready to install Asterisk. We should *not* run an X Server or any windowing software on our Asterisk machine, as the resources it consumes are almost guaranteed to delay our voice processing, and therefore negatively impact our sound quality. So you may save a little time and disk space by choosing not to install any such front end.

One note here: we should prepare to manage our server. We must keep in mind that we will not be able to rely on graphical tools on the server to manage users, file systems, and other aspects of the day-to-day maintenance all systems will need. Unless particularly comfortable with command-line configuration, you should probably consider installing a web-based set of tools to configure Linux, such as Webmin, available from `www.webmin.com`. The graphical configuration options for Asterisk that are available are mostly web based, so we may at some point decide to install these under a web server too, to enable graphical configuration.

Obtaining the Source Files

The very first step we must undertake is to obtain the source files. Two major versions of Asterisk are available: a development version and a stable version. Since the development version may not even compile at any given time, we will be using the stable version. We will be focusing on the version 1.0 branch.

When obtaining the source code, we have two major choices. We can either download the latest version via FTP from `ftp://ftp.asterisk.org/pub/asterisk/`, or use svn to obtain the latest stable release. The maintainers of Asterisk have been doing a good job of keeping the stable releases available on the FTP servers, so we will use this method.

The commands we issue to download Asterisk's source files are:

```
# cd /usr/src
# wget ftp://ftp.digium.com/pub/asterisk/asterisk-1.2.1.tar.gz
# wget ftp://ftp.digium.com/pub/asterisk/asterisk-addons-1.2.1.tar.gz
# wget ftp://ftp.digium.com/pub/asterisk/asterisk-sounds-1.2.1.tar.gz
# wget http://ftp.digium.com/pub/zaptel/zaptel-1.2.1.tar.gz
# wget http://ftp.digium.com/pub/libpri/libpri-1.2.1.tar.gz
```

The download may take anywhere from about one minute on an extremely fast connection to a couple of hours on a slow connection. When the download is complete, we will need to unpack the tarballs. We should also create a link by typing `ln -s /usr/src/asterisk-1.2.1 /usr/src/asterisk`. This will ensure the "addons" package compiles correctly.

In the next three sections, we'll compile and install the source distributions we've just downloaded. Note that we should install Zaptel first, then libpri, and finally Asterisk.

Installing Zaptel

The Zaptel sources are contained in `/usr/src/zaptel`. Type the following to install:

```
# cd /usr/src/zaptel
# make clean; make install
```

This will take about one to two minutes, depending on the speed of your machine. When it is finished, it should drop us back at the command prompt. If the last message states that there is a failure, we will have to do some detective work to determine the cause. The most common issues experienced will be resolved by meeting the dependencies listed earlier in this chapter.

Zaptel, containing the Zapata drivers created for Asterisk, is necessary to use Digium's telephony hardware, but also includes a number of libraries that Asterisk depends on, whether we use Digium's hardware or not.

If we want to have Asterisk start up at boot time, we should issue the command:

```
# make config
```

This command creates a script to insert the Zaptel module in the kernel and run `ztconfig` at boot time. In Red Hat, this script is then copied into `/etc/init.d` and configured to run at boot time for the current run level (which should be 3).

Installing libpri

Next, we will compile and install the sources contained in `/usr/src/libpri`. We do this by typing:

```
# cd /usr/src/libpri
# make clean; make install
```

This process should take less than a minute. Again, we know it is complete when we are dropped back at the command prompt.

Libpri provides the libraries required for using Primary Rate ISDN (PRI) trunks, as well as a number of other telephony interfaces. Even if we do not have a PRI line at this time, it is a good idea to install it, as it will not create any conflicts.

Parts of the Asterisk code depend on the libraries included in the libpri package. Therefore, any time we install libpri, we should recompile Asterisk.

Installing Asterisk

Now, it is time to actually install Asterisk, contained in `/usr/src/asterisk`, like so:

```
# cd /usr/src/asterisk
# make clean; make install
```

This installs the Asterisk PBX's runtimes and some utilities, as well as libraries. This creates the actual PBX, which may depend on (i.e. use) the components we installed earlier.

At this point, it is probably wise to install some sample configuration files so that we can acclimatize ourselves to Asterisk's structure. This is done by running:

```
# make samples
```

This creates a sample `zaptel.conf` in `/etc`, and sample configuration files in `/etc/asterisk`. When we change directories to `/etc/asterisk`, we should see the following files:

- `adsi.conf`: This file contains the configuration for Analog Display Services Interface, or ADSI for short.
- `adtranvofr.conf`: This file contains the configuration for using Adtran's Voice over Frame Relay.
- `agents.conf`: This file contains the configuration for using agents, like in a call center. This allows us to define agents and assign them IDs and passwords.
- `alarmreceiver.conf`: This file configures the alarm receiver application. We will not be changing the values from their default settings.
- `alsa.conf`: This file contains configuration variables for the console's sound card. We will not be using this.
- `asterisk.adsi`: Contains Asterisk's default ADSI script. This will be executed from the telephone if we use ADSI hardware.
- `asterisk.conf`: This file sets certain variables for Asterisk's use, most of which we will not need to change. It basically tells Asterisk where to look for certain files and executable programs.
- `cdr_manager.conf`: This file configures CDR for Call Management.
- `cdr_odbc.conf`: This is the configuration file for using an ODBC database connection to store our Call Detail Records (CDRs).
- `cdr_pgsql.conf`: This configuration file allows us to use a PostgreSQL database to store our CDR records.
- `cdr_tds.conf`: This is the configuration file for using FreeTDS, allowing connections to Microsoft SQL and Sybase.

- `enum.conf`: This file configures the use of ENUM, which allows us to resolve telephone numbers over DNS, thereby allowing us to route calls to an IP instead of going over the Public Switched Telephone Network (PSTN).
- `extconfig.conf`: With this file, we can choose to load our queues via the database engine.
- `extensions.conf`: This file configures the behavior of Asterisk. We will be working with this file extensively.
- `features.conf`: This file contains options for call parking, as well as a few miscellaneous features, such as the pickup extension, used for picking calls in each pickup group.
- `festival.conf`: This file sets parameters for Festival, which is an open-source program that allows our server to speak text. This is completely optional, and we will not go into configuring this, as Asterisk already includes recordings of the phrases we will need our server to say.
- `iax.conf`: This file configures our Voice over IP (VoIP) conversations using the Inter-Asterisk Exchange protocol, or IAX.
- `iaxprov.conf`: This allows for simple provisioning of Digium's S101I, also known as an IAXy.
- `indications.conf`: This is where we configure certain behaviors of our phone system, such as ring cadences and tones, enabling us to provide the sounds our users are used to, regardless of what country they are from. We can also mimic their previous phone system.
- `logger.conf`: This file sets up the type of logging we will be using. The defaults work for most people.
- `manager.conf`: This file configures remote access to the Asterisk Call Manager. This will be of utmost importance when we discuss Graphical User Interfaces (GUIs).
- `meetme.conf`: This configuration file sets up simple conference rooms. We can optionally define passwords for the conferences, too.
- `mgcp.conf`: This file configures Media Gateway Control Protocol, or MGCP. This is a protocol used by some VoIP hardware, mainly from Cisco.
- `modem.conf`: This file sets certain variables to allow us to use selected modems with Asterisk. Please note that not very many modems are supported, and as most modems are only half-duplex, they will not perform very well.
- `modules.conf`: This configuration file selects which Asterisk modules will be started up. We can enable or disable features of our PBX by changing configuration parameters here.

- `musiconhold.conf`: This configuration file creates Music On Hold (MOH) instances and defines what music they will play. At this time, it only supports playing MP3s, and only if we installed mpg123.
- `osp.conf`: We can configure the Open Settlement Protocol subsystem of Asterisk with this file.
- `oss.conf`: This configuration is much like `alsa.conf`, and we will not be using it.
- `phone.conf`: This file allows us to use some Linux telephony interfaces, such as the linejack by Quicknet. We will be focusing on Digium's hardware offerings instead.
- `privacy.conf`: This file allows us to configure privacy options.
- `queues.conf`: This configuration file allows us to create queues for callers to go through, allowing us to handle burst call volumes in an intelligent way. We can also create an escape to allow callers to dial their way out of the line.
- `res_config_odbc.conf`: This file sets the configuration for storing our settings in an ODBC database.
- `res_odbc.conf`: This is another piece of the configuration for storing our settings in an ODBC database.
- `rpt.conf`: This file allows us to use a radio repeater.
- `rtp.conf`: This configuration file sets the ports to use for Real-Time Protocol, or RTP. Note that the numbers listed are UDP ports.
- `sip.conf`: This configuration file defines Session Initiation Protocol (SIP) users and their options. We can also set global options for SIP, such as what address to bind to, what port to use, and what timeouts we are going to impose. SIP is a different protocol for Voice over IP.
- `skinny.conf`: This file configures the skinny VoIP protocol, which is used by many of the Cisco phones.
- `telecordia-1.adsi`: This is another sample ADSI script.
- `voicemail.conf`: This configuration file creates voicemail users and some global options for the Comedian Mail, Asterisk's voicemail system.
- `vpb.conf`: This file configures VoiceTronix hardware. We will be focusing on Digium's hardware offerings.
- `zapata.conf`: This file configures Zapata telephony interface settings. We will be using this to configure Digium's hardware offerings. Digium's hardware is what allows us to communicate with the PSTN.

As you can see, there are quite a few configuration files. Any particular installation of Asterisk may only use a few of these files, but they are all included so that we have the flexibility to use different features and may expand our services easily.

If we wish for Asterisk to start at boot time, we can configure it to do so by typing:

```
# make config
```

This will create a script to start the Asterisk PBX after the Zaptel startup procedures have been completed. In Red Hat Linux, this script is placed in `/etc/init.d` and set to execute when entering the run level we are currently in (which should be 3).

Getting to Know Asterisk

Now that we have installed Asterisk, there are some basic behaviors of Asterisk that we need to explore.

First, the major configuration files that need to be modified are in `/etc/asterisk`, with the exception of `zaptel.conf`, which is in `/etc`. Each file that ends in `.conf` is a configuration file, which sets parameters for some specific part of Asterisk, as described in the previous section.

The layout of these configuration files is generally simple. Most configuration files will have a variable name, followed by `=>` and its value. For instance, if there were a variable called `cat`, which was to be set equal to Garfield, it would look like this:

```
cat => Garfield
```

In most configuration files, the variables stay set until they are either undefined or set to a new value. You must be careful in what order you set variables, as they may have inherited a different setting than you anticipate. Therefore I suggest we set all needed values for every single instance, in case we make changes in the future that would break a set of lines in our configuration. This will make more sense later.

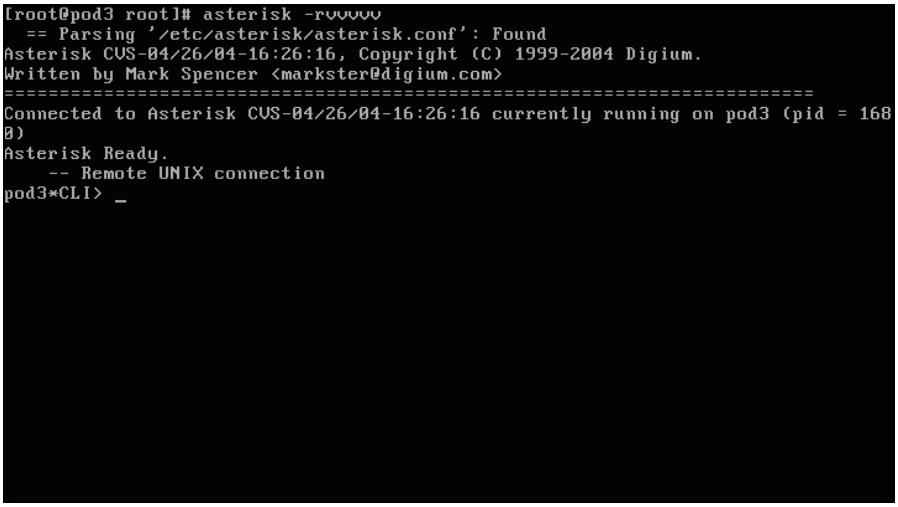
We should probably take a few minutes and look through all of the `.conf` files in `/etc/asterisk`. Most of the files are commented pretty well. We will be going through the key files step by step in the next chapter, but having a general knowledge of what is in there might help us.

Next, how do we start Asterisk? It's pretty simple, really. If you set Asterisk to start at boot time, you can reboot. If not, the command to start asterisk is `asterisk`. When starting Asterisk, there are a number of command-line arguments we can specify. The most commonly used are `-c`, which gives us a console connection, and `-v`, which gives us a verbose output, with more 'v's giving us more information about activities, status, and errors. When I start Asterisk from the command-line, I usually use:

```
# asterisk -cvvvvv
```

which gives me a console connection with plenty of debugging information.

If Asterisk starts at boot time, we can reconnect to the Asterisk console by typing `asterisk -r`, and we can specify a level of verbosity as above. So, when I reconnect, I often use `asterisk -rvvvvv`.


[root@pod3 root]# asterisk -rvvvvv
== Parsing '/etc/asterisk/asterisk.conf': Found
Asterisk CVS-04/26/04-16:26:16, Copyright (C) 1999-2004 Digium.
Written by Mark Spencer <markster@digium.com>
Connected to Asterisk CVS-04/26/04-16:26:16 currently running on pod3 (pid = 1680)
Asterisk Ready.
-- Remote UNIX connection
pod3*CLI>

Although the Asterisk console may not look like much, it is a wonderful tool for checking on the status of Asterisk, as well as diagnosing problems. We can type `help` to get a list of commands.

One very useful ability of the console is to issue a reload. This is done by:

```
CLI> reload
```

This command will re-parse the configuration files and update the changes in most of the modules of Asterisk.

There are some parts of Asterisk that require a restart in order to reflect changes. To restart Asterisk, we first must choose when we will be restarting. We have three main choices:

1. `now`: This option stops all calls in progress, immediately stops Asterisk and starts it again.
2. `gracefully`: This option does not stop calls in progress, but does not allow any new calls to be started. When all calls that are in progress are completed, the server will restart. Be careful, as a hung channel will basically disable your server.
3. `when convenient`: This is my favorite option. This option does not end any calls in progress, and allows new calls to start as usual. When there are no calls in progress, the server will restart. This means that incoming and outgoing calls are not interrupted except for the short period of time in which Asterisk is actually restarting. This would not work, of course, on high-load servers, as there may never be a time when there are no calls in progress, and it will also not work if a channel is "stuck", meaning the server thinks it has a call in progress, but it does not.

We restart Asterisk thus:

```
CLI> restart <choice>
```

For instance, if we wanted to restart now, we would type:

```
CLI> restart now
```

Now is a good time to play around with the console interface. Get comfortable with it. Experiment with the *Tab* key to auto-complete commands, and try different verbosity levels to see what information is displayed for each.

I cannot stress enough that now is the time to play with the server. If it is going to be broken by a mistake, it is better that the mistake be made before calls are going through it. If Asterisk stops working, we can go back to the beginning of this chapter and reinstall with minimal time.

Summary

In this chapter, we installed Asterisk and two packages that it requires, Zaptel and libpri. The configuration files that we use to set up various aspects of our PBX system have been introduced, as has the Asterisk console. Before our system is ready for use, we need to configure Asterisk for our hardware and other operational requirements, which we'll do in the next chapter.

PACKT
PUBLISHING

[PACKT PUBLISHING]

4
Configuring Asterisk

So far, we have decided to use Asterisk to meet our needs, created a plan, prepared a server, and installed Asterisk and its supporting libraries. Now we have come to the more artistic part of any open-source solution: configuration. We get to choose how to use Asterisk's power and flexibility to meet our real-world needs.

While the order in which we proceed makes sense, it is not necessary that we follow it precisely. We can configure the pieces in any order we want. The only issue we may encounter is that if we have not completely configured one part of Asterisk, the PBX may not start, or may start without full capabilities. This is not a real problem, as we are still configuring and will be testing our PBX later.

Thus, if we become unsure about how to proceed in one section, it may be best to move on, and configure the next part. Then, we can go back to where we left off. Often we need a little bit of time to digest some information.

What we will now do is step through the different technologies we're going to use, and the configuration files that we need to modify to follow the plan that we laid out in Chapter 2. Be sure to have the planning worksheets handy so that we are sure what we need to do.

> Before modifying the configuration files, we should make a backup copy of each.

We can create a backup copy of the stock configuration files by changing to the `/etc/asterisk` directory and making a copy of the files. For instance, if we wanted to make a copy of the `zapata.conf` file and call the copy `default-zapata.conf`, we would type:

```
$ cp zapata.conf default-zapata.conf
```

We could proceed to copy each other file in the `/etc/asterisk` directory in this manner, although if we overlook this backup step, the originals are of course available in Asterisk's source, under the `/usr/src/configs` subdirectory. By saving a copy of the default file, we can make any changes we wish to the configuration files without having to worry about trying to remember what the default file contained. If we have made extensive configuration changes ourselves we may want to be sure there is a backup copy of our modifications before we perform any other extensive changes.

Zaptel Interfaces

For this section, you need the details of the analog lines and terminals that you set out in Chapter 2. For our Zaptel interfaces, we will be modifying two configuration files, `/etc/zaptel.conf` and `/etc/asterisk/zapata.conf`.

zaptel.conf

As we know, the `zaptel.conf` file is in `/etc`. We can modify it in the text editor of our choosing.

As we make changes to this file, we will have to force the Zaptel drivers to reread the configuration files to detect the changes. If our system is configured to start the Zaptel hardware at boot time, we can accomplish this by running:

```
$ /etc/init.d/zaptel stop
$ /etc/init.d/zaptel start
```

If, however, we elected not to start Zaptel interfaces at boot time, we can implement our changes as we go by running:

```
$ ztcfg
```

To get more information, it is often helpful for us to use verbose flags. The more 'v's we use, the more verbose the output will be. So, we may wish to use:

```
$ ztcfg -vvv
```

> Remember: Changes to the `zaptel.conf` file will not take effect until we have restarted the drivers.

There are a couple of global options that can be set in the `zaptel.conf` file. First, there's `loadzone`, which defines country-specific preferences, such as what pitch a dial tone should have. Also, there's `defaultzone`, which tells Zaptel which zone to use as default if none is specified. For each of these, the value is the two-digit country code. Supposing we are in the United States, our `zaptel.conf` file could contain:

```
loadzone = us
defaultzone = us
```

Asterisk provides a number of defaults and we can configure additional ones in the `/etc/asterisk/indications.conf` file. Members of the Asterisk community have also contributed sections that we can add for particular countries, which can be found by using a search engine, as the Asterisk Users list is archived.

In this file, we have two major classes of devices. First, we have lines. This refers to all of the connecting links we have to the Public Switched Telephony Network (PSTN). Then, we have all of our terminals, which refer to modems, telephone sets, cordless phones, fax machines, war dialers, or any other analog devices we may wish to use.

Lines

Each line can have many different types, and which type we use depends on what services our telephone company is providing. In the United States, two common choices are Primary Rate ISDN (PRI) and Plain Old Telephone System (POTS) lines, also known as analog lines.

In previous chapters, we made a list of all the lines we have coming into our building. We also selected which lines we would be tying to Asterisk. Since we are only discussing the configuration of a PBX, we will ignore any lines not tied to Asterisk; however, you should keep the documentation for your future reference.

In the `zaptel.conf` file, we need to identify the signaling we will be using. Since these are lines to the PSTN, we will be using FXS signaling. If you remember, we learned that FXO devices are what we need to connect to the PSTN. But here we define the signaling Asterisk is to use, not the type of device. The signaling will be exactly opposite from the device type.

For each Digium channel (i.e. port on a TDM400P or X100P), we have to define the following:

```
fxsks=1
```

Note that we are defining three very crucial pieces of information here. First we specify signaling, which is FXS since this interface is an FXO device. Secondly, we designate the protocol. This depends on the phone line that we use. Usually, we can use Koolstart (ks). Other options are Loopstart (ls) and Groundstart (gs). Finally, we set the channel number to 1. This depends on the order in which our system detects the modules. Thankfully, if we configure the values incorrectly and run `ztcfg`, we will see an error message like:

```
$ ztcfg
ZT_CHANCONFIG failed on channel 1: Invalid argument (22)
Did you forget that FXS interfaces are configured with FXO signalling
and that FXO interfaces use FXS signalling?
```

As we can divine from the error message, this is not an uncommon problem. It is no big deal, we just know that our system believes that channel 1 has the opposite type of signaling to what we defined. We can go back into our `zaptel.conf` and switch the signaling. This can help us determine the order in which our system is loading the Zaptel interfaces if we are unsure.

If we have a Primary Rate ISDN (PRI) or a T1 coming in, either from the telephone company or a channel bank, then our configuration will be a bit different. We define each T1 as being a span. For each of these spans, we need to specify the following details:

- **Span number**: this is an arbitrary number that we assign to each T1. The T1 connected to the first port should be 1, the second T1 connect should be 2, and so on.
- **Timing**: this is an integer that represents the order in which we rely on it for synchronizing our timing. 0 means that we will not use this span for synchronization, 1 indicates it is our primary source, 2 indicates that it is our first backup timing source, 3 indicates it is our second backup timing source, and so on.
- **Line Build Out** (LBO): this integer represents the distance of the cable. Valid options are:
    1. 0 db or 0-133 feet (if you are unsure, try this)
    2. 133-266 feet
    3. 266-399 feet
    4. 399-533 feet
    5. 533-655 feet
    6. -7.5 db
    7. -15 db
    8. -22.5 db
- **Framing**: this information is specific to our connection. While we should be able to get this information from our telephony provider, many users have reported difficulty finding anybody at the telephone company willing and able to provide this information. Therefore it is convenient that there are only a few options.

  For a T1, the valid options are:

    - `d4`, also known as `superframe` or `sf`
    - `esf`

  For an E1, the possibilities are:

    - `cas`
    - `css`
- **Coding**: this information is also specific to our particular line, and again should be available from our connection provider.

  For a T1, it could be:

    - ami
    - b8zs

For an E1, we could have:

- `ami`
- `hdb3`
- `crc4`: added to `ami` or `hdb3` to enable CRC-4 checking

So, supposing we have a PRI in the US, using ESF framing and B8ZS coding, we would have:

```
span=1,1,0,esf,b8zs
```

Now that we have defined the span, we must configure the channels for use. To do this, we use statements of the form `<device>=<channel>`. Here `<channel>` can be a single number representing a specified channel, a comma-separated list of channels, or a range, using a hyphen between the first and last channels. For a T1, some common options for `<device>` are:

- `e&m`: This will work irrespective of implementation, such as Immediate, Wink, or Feature Group D.
- `fxsls`: this is used with many channel banks, referred to as "loopstart".
- `fxsgs`: this is used with many channel banks as well, referred to as "groundstart".
- `unused`: this tells Zaptel to ignore the channel, for instance when only a fractional T1 is delivered by the telephone company.
- `bchan`: Also known as "indclear", this tells Zaptel not to perform any conversion.
- `dchan`: Also known as "fcshdlc", this tells Zaptel to perform HDLC encoding and decoding on the bundle, and send through this device.

A full list of the possible devices is contained in the sample configuration file, or in `/usr/src/zaptel/zaptel.conf.sample`. Continuing with our example of a PRI in the US, we would have:

```
bchan=1-23
dchan=24
```

If we had two identical PRI lines coming in, the entire `zaptel.conf` file would look like:

```
# First incoming PRI
span=1,1,0,esf,b8zs
bchan=1-23
dchan=24

# Second incoming PRI
span=2,2,0,esf,b8zs
bchan=25-47
dchan=48

loadzone=us
defaultzone=us
```

The configuration for a channel bank will be very similar. Supposing we had a single channel bank, using ESF framing and B8ZS coding, and groundstart lines on channels 49, 50, 51, and 52, we would have:

```
# Channel bank to PSTN
span=3,0,0,esf,b8zs
fxsks=49-52
```

Terminals

Just as our configuration of the lines depends on the capabilities of our telephone provider, as we configure our terminals, we must keep our equipment in mind. Our settings in `/etc/zaptel.conf` for our terminal devices will be directly related to the type of equipment we are using.

As our analog terminal devices will be using an FXS device to connect, Asterisk must use FXO signaling to communicate. Therefore, if we have a TDM400P with FXS modules, we would use fxoks signaling. Assuming this is our second port to load during `ztcfg`, the entry in `zaptel.conf` will be:

```
fxoks=2
```

We can also connect our Asterisk server to a channel bank or other phone system. This is done just as it was for defining lines. We must first designate a span (in the same format as before) and then configure the individual channels.

For instance, suppose we have a channel bank that has 4 FXO ports and 20 FXS ports. Imagine it is using ESF framing and B8ZS coding. Thus, we could have:

```
# Channel bank to PSTN and Terminal Devices
span=3,0,0,esf,b8zs
fxsks=49-52
fxoks=53-72
```

No problem, right? Remember: as we make mistakes, when we run `ztcfg`, we will get some useful error messages. The more verbose the output, the more likely we are to get a hint as to what is causing any problems we may experience.

zapata.conf

Now that we have configured `/etc/zaptel.conf`, we have our telephony devices starting. If our FXO or FXS device is not starting yet, we need to go back to the previous section, as this is one of the few configuration files that depend on the success of another. Until we have our interfaces loading, we will not have much success in `/etc/asterisk/zapata.conf`.

This configuration file is read by Asterisk. Therefore, to read changes made to this file, we can issue a `reload` in the Asterisk console. Zaptel will NOT have to be restarted to apply any changes we make in `zapata.conf`.

At the top of our file, we will see a [channels] section. In fact, that is the only section we will have. At the beginning, we will set certain characteristics that we want to be consistent between all of our lines and terminals. We must exercise great care when working with this file. When we set any variable, it stays in force during all of the later channel declarations until we set it otherwise. For those of us who program, it is much like a switch statement without breaks at the end of cases: sometimes useful, always dangerous!

Each of these settings that we discuss can be reset at any point in the file, whether for incoming or outgoing interfaces. As we reset it while defining one interface, all of the next lines will have the same setting. Therefore, it behooves us to group lines of similar configuration together so that we won't need to reset variables as often.

Not every setting will be needed for every channel. In fact, most installations probably don't even use half of the settings available. However, we should go over all of the possibilities, as every installation is a little bit different, and one of these settings might be just what we need!

As there are so many options that must be set, we will go through them in list format.

- `Language`: This is the default language to use. The default is "`en`" for English.
- `context`: This is how we tell Asterisk which context to put new calls in. The default value is "default".
- `switchtype`: Used only for Primary Rate ISDN, the valid options are:
  - `4ess`: AT&T's 4ESS protocol
  - `5ess`: Lucent's 5ESS protocol
  - `dms100`: Nortel DMS100
  - `euroisdn` - EuroISDN
  - `national`: National ISDN 2 (default)
  - `ni1`: Old National ISDN 1
- `pridialplan`: This setting is only occasionally used for PRI connections. The options are:
  - `unknown`
  - `private`
  - `local`
  - `national`
  - `international`
- `overlapdial`: This setting decides whether or not to send overlapping digits while dialing. Valid options are "yes" and "no".
- `signalling`: This setting chooses the signaling method. Valid options are:
  - `em`: E&M signaling

    - `em_w`: E&M Wink
    - `featd`: Feature Group D, Adtran style
    - `featdmf`: Feature Group D, US
    - `featb`: Feature Group B, US
    - `fxY-ZZ`, where `Y` can be `o` or `s` (this will be the same as in `zaptel.conf`), and `ZZ` can be `ks` for Koolstart, `gs` for Groundstart, or `ls` for Loopstart
    - `pri_cpe`: PRI signaling, Customer Premises Equipment (CPE) side
    - `pri_net`: PRI signaling, Network side
    - `sf`: SF Signaling
    - `sf_w`: SF Wink
    - `sf_featd`: SF Feature Group D, Adtran style
    - `sf_featdmf`: SF Feature Group D, US
    - `sf_featb`: SF Feature Group B
- `prewink`: a prewink time, defaults to 50 ms.
- `preflash`: a preflash time, defaults to 50 ms.
- `wink`: wink time, defaults to 150 ms, while Atlas uses 250 ms.
- `flash`: flash time, defaults to 750 ms.
- `start`: start time, defaults to 1500 ms.
- `rxwink`: receiver wink time, defaults to 300 ms.
- `rxflash`: receiver flash time, defaults to 1250 ms.
- `debounce`: debounce, defaults to 600 ms.
- `usedistinctiveringdetection`: set this to "yes" for our FXO interface if our phone company sends us distinctive ring.
- `usecallerid`: set this to "yes" if we wish to use caller ID. One note: this may cause a delay between rings and the pickup of Asterisk on calls, as Asterisk will have to wait for caller ID information to be available. If our installation cannot handle this, then we will have to set it to "no".
- `hidecallerid`: set this to "yes" if we will want to mask our caller ID on outgoing phone calls.
- `callwaiting`: set this to "yes" to enable call waiting on FXO devices.
- `restrictcid`: this sends caller ID as ANI (Automatic Number Identification) only, and not available for the user.

- `usecallingpres`: this toggles whether we want to use the caller ID presentation for outgoing calls that the calling switch is sending.
- `callwaitingcallerid`: this sets if we support caller ID for call waiting.
- `threewaycalling`: this sets if we support three-way calling.
- `transfer`: here we can decide if we're supporting flash-hook transfers. The use of this feature requires three-way calling.
- `cancallforward`: set this to "yes" if we want to be able to forward calls.
- `callreturn`: whether or not to support *69 for call return.
- `mailbox`: here we set the voicemail number of the mailbox. If this is set to a valid voicemail number, and that account has a new message, then our user will hear a stutter dialtone when he or she picks up the phone. If there is more than just one context for voicemail, we can specify it by `user@context`.
- `echocancel`: this variable can be set to "yes", "no", or a number, defining how many taps to cancellation (needs to be a power of 2).
- `echocancelwhenbridged`: this sets whether we want to cancel echo when the circuit is purely Time Division Multiplexing (TDM). Usually, echo canceling is unnecessary; however, the sample configuration has it set to "yes".
- `echotraining`: this variable can be "yes", "no", or "800". It has been reported on the User's list that a setting of 800 fixes most of the echo experienced on TDM400P and X100P interfaces.
- `relaxdtmf`: this setting can help if Asterisk keeps detecting voice as DTMF (digits).
- Gain settings:
  - `txgain`: in dB, sets the TX gain, default of 0.0
  - `rxgain`: in db, sets the RX gain, default of 0.0
- `group`: we can assign groups to our lines. This allows rollover when making outgoing calls.
- `callgroup`: this is a ring group.
- pickupgroup: this is a pickup group. If another phone in your pickup group rings, you can pick it up by dialing *8#.
- `immediate`: this sets if we want calls picked up immediately, or if we want to provide dialtone. Default is "no", meaning we will provide dialtone, read digits, and complete the call.
- `callerid`: we can set the caller ID number to "asreceived" or override it with a specific number. "asreceived" only makes sense on trunk lines.

- `amaflags`: affects the recording of Call Detail Records (CDRs). This variable can be set to "default", "omit", "billing", or "documentation".
- `accountcode`: we can tie channels to account codes for billing purposes.
- `adsi`: we can set this to "yes" if we have ADSI-compatible handsets.
- `busydetect`: this can be set to "yes" to try to find if lines are hung up or busy.
- `busycount`: this variable sets how many busy tones to hear before hanging up a channel. The default value is 4, but increasing this may prevent some seemingly random hang-ups.
- `callprogress`: this variable sets whether or not to use the experimental call progress detection algorithms. One note of warning: this tends to cause random hang-ups. It is probably best to wait until this is more stable to use.
- `progzone`: used in conjunction with `callprogress`. Set this to "us" for the United States.
- `musiconhold`: sets which class of music on hold to use.
- PRI idle extension: this group of settings can be used to more effectively use channels on a PRI line. It is often used to multilink through PPP.
- `idledial`: sets the extension to dial from the idle line.
- `idleext`: sets the extension to dump the idle line to.
- `minunused`: sets minimum number of channels to leave unused.
- `minidle`: sets minimum number of channels to leave in the idle extension.
- `jitterbuffers`: default is 4. This is designed to smooth out jitter.
- `cadence`: defining custom ring cadences. Defining any here will cause the default cadences to be turned off.
- `channel`: this can be a channel or range of channels. These channels must match those defined in `/etc/zaptel.conf`.

While there are many options available, we will only need a few of them. And most of them, we can set once and not need to set them again. Usually, the defaults work fine, but at least we know that if we're unhappy with the way our phone systems acts, we can easily configure it to perform the way we wish it to.

For `/etc/asterisk/zapata.conf`, we have the two major divisions of devices, as we had in `zaptel.conf`, namely lines and terminals. The definitions are the same as before.

Lines

Once we have set the options above, all we have to do is define the channel number. It is best to set the signaling method within sight of the channel definition so that problems are easier to debug. Supposing we have an FXO device on channel 1, which we want to call group 1, we would have the following lines in our `/etc/asterisk/zapata.conf` file:

```
signalling=fxs_ks
context=default
group=1
channel=1
```

Grouping these four lines close together in the file will make it easier to troubleshoot any problems in the future.

Suppose we have two PRIs, as we did in the example in the lines section of `zaptel.conf` file. Our channel definitions could look like:

```
; incoming PRI
callerid=asreceived
context=default
switchtype=dms100
signalling=pri_cpe
group=2
channel=>1-23
channel=>25-47
```

By putting both of the incoming PRIs together, we saved ourselves some typing. As long as we're happy with both trunks being in the same group, there is no reason why we should have to redefine the variables.

One note on security: we need to be careful what context we put our incoming calls into. If we place them in a context that can dial long distance, then people can relay telephone calls through our server.

Terminals

When defining the channels that our terminal devices connect to, we need to remember to take into consideration more specific details about the uses of the channel. For instance, analog handsets will be unable to send caller ID information, so setting `callerid=asreceived` does not make much sense.

Furthermore, here is where we have to remember if we are using ADSI devices. While it may be tempting for us to turn on `adsi=yes` for all lines, it tends to make users angry when they hear some of the high-pitched beeps that sometimes are emitted during conversations. Therefore, we should only enable ADSI on phones that are ADSI compatible.

An example configuration for an FXS device on channel 2 might be:

```
signalling=fxo_ks
context=longdistance
callerid="My Name Here"<(850) 555-5555>
adsi=no
callgroup=1
pickupgroup=1
channel=2
```

As you can see, this phone does not support ADSI, has the correct caller ID set, and has been placed in the call group 1, so that the user can pick up any calls for terminals in the same group. Also, we need to take notice that the line `channel=2` appears last.

We can repeat this process for all of the channels we need to define. Lines through channel banks will be configured the same. In keeping with our previous example, suppose we have a channel bank with 4 FXO devices, and 20 FXS devices. The section would look like:

```
; channel bank configuration
; fxo devices
signalling=fxs_ks
context=default
group=1
callerid=asreceived
channel=49-52
;fxs devices
signalling=fxo_ks
context=longdistance
group=
callerid="John"<1234>
channel=53
callerid="Jacob"<2345>
channel=54
callerid="Jingleheimer"<3456>
channel=55
callerid="Smith"<4567>
channel=56
callerid="My Company"<(555) 555-5555>
channel=57-72
```

Notice that I have not given a value for `group` in the `longdistance` context. This moves these channels to be outside a group. In this instance, I don't feel using a group number would be appropriate, as each phone may be for different functions, etc.

As you can see, the only different piece of information for each of the lines in the channel bank is the caller ID. This makes it very convenient to define all of the channels. We now have plenty of information to configure all of our Zaptel terminals.

SIP Interfaces

Session Initiation Protocol, or SIP, is a standardized Voice over IP (VoIP) protocol. This protocol relies heavily on the Real-time Transport Protocol (RTP), which uses UDP ports in the TCP/IP stack. It presents addresses in much the same format as email, as `user@domain`.

We configure this protocol by editing `/etc/asterisk/sip.conf`. This file has a number of settings in a `[general]` section, followed our definitions of users.

There is a whole host of options that we can set. These options include:

- `context`: sets the default context for calls coming into the server. These calls can be from our users, or if we are connected to the Internet, they can be from anywhere. Just to be on the safe side, we should not set this to be a context that can call long distance. The default is "default".

- `realm`: sets the realm of the server. As we discussed earlier, the calls are addressed like email, in that the format is `user@domain`. This variable is how we set the domain part. This could be your host name or a domain name. If we enter nothing, it will work, and will set our realm to "asterisk", but we really should set this to our domain name.
- `port`: sets the UDP port to listen on for connections. The default is 5060, and we shouldn't change this unless we have a REALLY good reason.
- `bindaddr`: specifies the IP address that we want the SIP service to bind to. Keep in mind that if a machine has multiple IP addresses, we can specify that it bind to all by using 0.0.0.0 as the `bindaddr`. Also keep in mind that if we ever change the IP of the server, we will have to adjust this variable, unless we use the 0.0.0.0 address. It may be a good idea to only bind where you expect SIP traffic, for added security.
- `srvlookup`: determines if DNS SRV lookups are enabled. SRV records in DNS are a way to allow other Internet users to point to our SIP server without having to know its host name. This also allows us to be able to change SIP servers without updating everybody's address. We can think of this in the same vein as a DNS MX record. We should set this to "yes".
- `pedantic`: if we have pingtel phones, we should set this to "yes". This enables pedantic checking for multiline formatted headers.
- `tos`: this is the Quality of Service (QoS) setting. We can choose to specify a numerical value, or use the keywords "lowdelay", "throughput", "reliability", "mincost", or "none". We should probably set this to "lowdelay", as users are typically pretty annoyed by pops, cracks, and other inconsistencies in voice conversations.
- `maxexpiry`: this is the maximum length, in seconds, that we allow incoming registrations to stay valid. We should set this to something reasonable. The default is 3600, meaning registrations will time out after one hour.
- `defaultexpiry`: this is the default length in seconds that we set for incoming registrations. The default is 120, or 2 minutes.
- `notifymimetype`: we can override the MIME type in SIP NOTIFY messages. We should not modify this unless we absolutely have to.
- `videosupport`: we can set this to "yes" to allow video support in SIP. Asterisk is currently known to support H.263 video from Linphones, Microsoft Windows Messenger, and the Wooksung WVP-2000 videophone.
- `musicclass`: here we can set the default music on hold class for all SIP calls. The default is "default".

- `amaflags`: we have the same choices here as for `amaflags` in `zapata.conf`: "default", "omit", "billing", or "documentation".
- `accountcode`: we can set the account code to use for calls from this SIP user. This can help with billing.
- `language`: this is the default language for SIP users. We can set this to the country where the server is, and then override for any users with a different language. The default is "en".
- `relaxdtmf`: just as in the `zapata.conf` file, we can relax the handling of DTMF in SIP. This can be useful if Asterisk is detecting digits pressed in the middle of voice conversations.
- `rtptimeout`: here we configure how long a period of inactivity will time out calls. We set this to the number of seconds the RTP stream has to have no activity before Asterisk will terminate the call. The default setting is 60.
- `rtpholdtimeout`: here we can configure the hold timeout, in seconds. We set this to a value greater than the `rtptimeout`. The default setting is 300.
- `externip`: here we set the external IP address of the Asterisk server. This is very useful for piercing through NAT and some firewalls. This address will be placed on outbound SIP messages.
- `localnet`: here we define all of the internal IP addresses. This tells Asterisk which SIP messages to use the external IP on, and which ones need the internal IP address.
- `register`: here we can tell Asterisk to register with a SIP provider or service. An example of this would be:
  `register => john:johnspassword@sipprovider.com.`
  Optionally, we can specify the port of the remote server to use (if it is different from the default 5060) and the extension to drop all calls into. If we were to specify these, and we wanted to use port 5061 and place all incoming calls into extension 9999, it would look like:
  `register => john:johnspassword@sipprovider.com:5061/9999`.
- `codecs`: now we must define what encoders/decoders (codecs) we will allow. This is done through a series of allow and disallow statements. First we should disallow all codecs by using the statement `disallow=all`. Then we can enable codecs one at a time by typing `allow=ulaw`, `allow=ilbc`, etc., to allow all of the codecs we wish to use. The order in which we enable the codecs will be the order in which Asterisk tries to negotiate them.

Now that we have configured the global options, we must define our users. As we do so, we can use most of the previous settings, as well as a few new options. One note: in the general section, the order of allowed codecs mattered; however, as we define our users, order will not guarantee the order of negotiation. The following is a list of the options available to us as we define users:

- `type`: there are three major types of users:
  - `user`: this connection is permitted to send calls to us.
  - `peer`: we are permitted to send calls to this connection.
  - `friend`: this connection is both a user and a peer.
- `username`: sets the username for authentication.
- `secret`: sets the password used for authentication.
- `md5secret`: MD5 hash of `<user>:asterisk:<secret>` for more secure authentication.
- `fromuser`: overrides caller ID, and is required by Free World Dialup (FWD).
- `callerid`: sets the caller ID. An example would be `callerid=My Name <1234>`. It's usually a good idea to set `fromuser` and `callerid` to the same string.
- `host`: sets the host address of the user. This can either be a static address or the keyword "dynamic".
- `defaultip`: used with `host=dynamic`. This sets the default IP for when the extension has not registered.
- `nat`: sets whether or not this SIP device is behind a NAT firewall
- `mailbox`: sets the mailboxes to check for messages for this user. This can be just the mailbox ID, or can be the mailbox ID, followed by `@contextname`.
- `qualify`: how many milliseconds the device can be unreachable before considering it as down.
- `canreinvite`: we should set this to "no" if one of the devices is behind a NAT. When two devices start having a conversation, they try to reinvite each other, thereby skipping the Asterisk server. This is good to keep the load down, as well as to lower the number of hops required in the network, but if Asterisk is how we get through the NAT, the reinvite may not work.
- `outgoinglimit`: we can set the maximum number of calls a device can take. Setting this to 1 will disable `callwaiting`. Note that this option and `incominglimit` below have been deprecated, and are commented out in the code. They could be re-enabled with a little effort by directly modifying the source; however, this is not recommended as doing so makes updating difficult, as well as causing strange issues where calls to SIP channels are not counted correctly.
- `incominglimit`: we can set the maximum number of calls a device can initiate at a time. Setting this to 1 will disable the ability to do three-way calls or transfers on some SIP phones. See note for `outgoinglimit` above.

- `dtmfmode`: this sets the DTMF mode. The choice we make depends on the hardware we're using. The valid options are "rfc2833", "info", and "inband".
- `callgroup` and `pickupgroup`: these can be set as they were in `zapata.conf`.
- Security: we can use the keywords "permit" and "deny" to provide some level of security. Order matters, as the last matching rule will be the one followed.
- `deny`: lists IP addresses to deny. If we issue a `deny=0.0.0.0/0.0.0.0`, then all attempts are denied. If we issue a `deny=192.168.1.0/255.255.255.0`, then all connection attempts from the class C space of 192.168.1.0 are denied.
- `permit`: lists IP addresses to permit. By default, all IP addresses are permitted. If we used deny 0.0.0.0, then we will now have to list all IPs and blocks of IPs that we should allow.

Now that we have gone through the options, let's look at some examples. Suppose we have already set all of the general options. We have three users: one for all incoming SIP calls, one for calling out of Free World Dialup (FWD), and one for a SIP handset.

```
[sip_incoming]
type=user

[FWD-out]
type=peer
secret=mypassword
username=myusername
fromuser=myusername
host=host.provider.com

[1000] ;this is the extension of the handset
type=friend
context=longdistance
username=1000
callerid=My Name <1000>
host=dynamic
defaultip=192.168.1.100
secret=2manysecrets
nat=no
canreinvite=yes
dtmfmode=info
outgoinglimit=1
incominglimit=2
mailbox=1000
disallow=all
allow=ulaw
amaflags=default
accountcode=company123
```

As you can see, the definitions can be as short as one line (assuming we defined our default context in the general section) or as long as we need them to be. Now we need to edit our SIP configuration file for our needs. Take a look at the terminal devices that we listed in Chapter 2 on which we selected SIP as our protocol. We now have enough knowledge to be able to create a user for each one of them.

IAX Interfaces

Asterisk provides another VoIP protocol, much like SIP, called Inter-Asterisk eXchange, or IAX. This protocol is easier to work with for many reasons, as we discussed in Chapter 2. For this section, we will need all of the terminal device details for which we selected IAX as the protocol.

Just as the name suggests, IAX is well-suited to connecting multiple Asterisk servers together. At this point in our work with Asterisk, this is not what we need, as we are limiting ourselves to one server. When we link multiple servers together, this feature will become useful.

Following the pattern established, the IAX protocol is configured in `/etc/asterisk/iax.conf`. Just as the SIP file, it has a number of general settings, followed by settings for each individual users.

First we will discuss options for the `[general]` section. They include:

- `port`: set the port to listen on. The default is 5036, which we should not change without a very good reason.
- `bindaddr`: set the IP address to bind to. By default, it will bind to 0.0.0.0, meaning all available IP addresses.
- `amaflags`: just as for SIP and Zaptel interfaces, we can set `amaflags` for IAX.
- `accountcode`: as for SIP and Zaptel interfaces, we can set an account code to be used for billing purposes.
- `language`: we can specify a language. If omitted, Asterisk will default to English.
- `bandwidth`: we can specify a level of bandwidth. This will automatically set which codecs can and cannot be used. The valid choices are "low", "medium", and "high".
- Codec selection: just as in `sip.conf`. We can disallow all codecs, and only allow the codecs that we want to permit. If we use `allow=all`, it is the same as using `bandwidth=high`.
- `jitterbuffer`: this allows us to configure our jitter buffer. We should leave it at `jitterbuffer=no` unless our network is unusually jittery because of the added latency that using buffers will typically create.

- `trunkfreq`: we can set the number of milliseconds between trunk messages. The default of 20 ms should be good enough for most installations.
- `register`: just as in `sip.conf`, we can register with a remote server. If our username is "johndoe", and our password is "foo", and we want to register with reallycoolhost.com, then the statement would look like `register=>johndoe:foo@reallycoolhost.com`.
- `tos`: here we set our TOS bits. Valid choices are "lowdelay", "throughput", "reliability", "mincost", and "none". Usually "lowdelay" is desirable.

Now that we have set our options in the general section, we need to define users. We can see that it is very similar to the way we configured SIP. One difference between the SIP and IAX protocols is that IAX will support RSA public/private key encryption, along with the MD5 and plaintext authentication methods. If we choose to use RSA authentication, then we must put the keys in `/var/lib/asterisk/keys/`. All public keys must end with `.pub`, while private keys will end with `.key`.

There are a few options that can be set per user entry, in addition to the settings mentioned above. They include:

- `type`: just as in SIP, we can have "user", "peer", or "friend".
- `auth`: can be "md5", "plaintext", or "rsa".
- `inkeys`: name of the key file or files to use, with a colon between each acceptable public key for RSA encrypted authentication.
- `outkey`: name of the private key file to use for RSA-encrypted authentication.
- `secret`: the password to use.
- `notransfer`: if set to "yes", this will disable IAX's ability to use native transfers.
- Security: much like the allow/disallow pair for codec selection, we can use permit/deny directives for IAX connections.
    - `deny`: list of all IP addresses and blocks to not allow access from. If we enter "0.0.0.0/0.0.0.0", then all access will be blocked.
    - `permit`: list of all IP addresses (as individual addresses or blocks) to permit access from. If we enter "192.168.1.100/255.255.255.255", then the user at 192.168.1.100 will be permitted access.
- `qualify`: we set this to "yes" when we want to make sure a user's device is connected before attempting to send calls to it.
- `trunk`: we set this to "yes" to use IAX trunking. Trunking refers to putting multiple phone calls in the same stream to save the overhead of packaging each conversation individually. If we will often have multiple conversations in progress with the same host (i.e. another Asterisk server uses this account), then it is a good idea to set `trunk=yes`.

Now we have enough information to create all of our IAX users, we will go through an example. Suppose we have a user who will be on extension 2000, with a password of 2000iscool. This is a hardware IAX phone, and as such, will be dynamic and be of the friend type. The user will be permitted to make long-distance telephone calls, and their telephone will send the correct caller ID name and number, and we will trust it. This user is only permitted to connect from our local subnet, which is the class C address of 192.168.1.0.

```
[2000]
type=friend
secret=2000iscool
host=dynamic
defaultip=192.168.1.200
callerid=asreceived
context=longdistance
deny=0.0.0.0/0.0.0.0
permit=192.168.1.1/255.255.255.0
```

We can now go through our listed devices and make similar entries for any of our terminal devices of the type of IAX.

Voicemail

Asterisk provides a voicemail program called Comedian Mail. Through `/etc/asterisk/voicemail.conf`, we can configure global options for our voicemail system as well as define different voicemail boxes.

In the configuration file, we first have our general options:

```
[general]
```

The first setting we get to decide on is how to write the voice files as they are recorded. The default is usually OK:

```
format=wav49|gsm|wav
```

Next, we get to set up email notification. Comedian mail will allow us to notify users of new messages via email, and optionally, we can attach the voice files directly to the message. This is the reason we select `wav49` as a format above, as most computers will be able to play the files. We choose the email return address and whether we will attach the voicemail to the email with `serveremail` and `attach` respectively. We can also choose the display name that the email comes from, by setting `fromstring`. If we need to, we can also override the email sending program, in the `mailcmd` variable.

```
serveremail=asterisk@mydomain.tld
fromstring=Asterisk PBX
attach=yes
mailcmd=/usr/bin/sendmail -t
```

Now we need to set some limits on messages. We will define `maxmessage`, which is the maximum length, in seconds, of a voicemail message. The variable `minmessage` is similar, in that it is the minimum length, in seconds, of a voicemail message. We also set `maxgreet`, which is the maximum length our users can record for a greeting, again measured in seconds. Through `maxsilence`, we set how many seconds of silence we accept before ending the recording. Thus, we could set:

```
maxmessage=180
minmessage=3
maxgreet=45
maxsilence=5
```

These are not the only variables for controlling the general behavior of Comedian Mail. We also have `skipms`, which defines the number of milliseconds to move ahead or back when fast forwarding and rewinding the message, `maxlogins`, which sets how many failed attempts users can have in one session before being disconnected, and `silencethreshold`, which allows us to configure what the system will consider to be silence. For the silence threshold, the smaller the number, the lower ambient noise will have to be before it is recognized as silence. To continue our example, we could have:

```
skipms=2000
maxlogins=3
silencethreshold=128
```

There are more options available for us to change the body of the email, notify external programs of new voicemail messages, change our character set, and a whole host of other options; however, the default settings should work in most installations. If they do not meet our needs, we can configure them at any time.

Now that we have configured the general options, we have the option of creating timezone messages. This is how we let Comedian Mail know what it should say when telling users about when messages arrived. For instance, if we lived in the Central timezone, we would have:

```
[zonemessages]
central=America/Chicago|'vm-received' Q 'digits/at' IMp
```

Each timezone is defined in `/usr/share/zoneinfo/`. Just find a city in the desired timezone to define the zone message.

Finally, we have to define actual mailboxes. We have to place each mailbox in a context. This context should be the same as the context that the users' extension will appear in, and if possible, the voicemail box should be the same as the extension of the user. The format for the configuration line is:

```
voicemailbox => password,username,emailaddress,pageraddress,options.
```

We can use one or more options for each user. Some of the options to be aware of are: `tz` to set the timezone, as defined in `[zonemessages]` above, `attach`, which tells Comedian Mail whether this particular user wants to have their voicemail file attached to the email, `saycid`, which can make Comedian Mail speak the Caller ID of the caller who left the message, and `operator`, which allows us to define whether the caller can press 0 to get an operator while leaving a message. Each option is separated by the pipe ("`|`") symbol.

So, if we wanted to have a normal voicemail box for extension 1000, with a password of 1234, and accepting all of the defaults, and not sending email, we would have a line like the following in `voicemail.conf`:

```
1000 => 1234,Example Mailbox
```

Now, suppose we wanted to set up a little more of an advanced example. Suppose Joe User, on extension 1001, with a password of 123456789, has an email address of `juser@domain.tld`, no pager, and lives in the Central timezone. He wants his voicemail message attached to his email, wants Comedian Mail to speak the Caller ID, and wants his callers to be able to press 0 for an operator. His voicemail box line would look like:

```
1001 => 123456789,Joe User,juser@domain.tld,,tz=central|attach=yes|
saycid=yes|operator=yes
```

Great! We have just configured a fully-functional voicemail box. In a similar fashion, we can now create all of the voicemail boxes we need.

Music On Hold

Using music on hold, Asterisk enables us to stream MP3 files to any handset or line. These streams are commonly used for Music on Hold and for the music played while people are waiting in a queue. Each stream is configured in `/etc/asterisk/musiconhold.conf`.

Asterisk gives us the flexibility of defining multiple instances of MOH, referred to as classes. Each class can use a different directory of MP3s and a different mode. For our purposes, we will only be using the mode called **files**. This mode allows us to use Asterisk's native players to stream music on hold.

Asterisk's definition of Music on Hold has change quite a bit for version 1.2; therefore, if we have some older configuration files from 1.0.x, we will need to update them for the new release.

Using Asterisk's native player also allows us to have our Music on Hold in various formats that Asterisk supports. Our PBX will determine the best format to play; therefore, transcoding may be avoided if we create our Music on Hold files in the format our channels are in.

There are two main directives we need to be aware of: mode and directory:

- **Mode** refers the the mode we previously discussed. We will be using files, meaning we will use Asterisk's native player.
- **Directory** simply points Asterisk to the directory holding the Music on Hold files.

If we choose to shuffle the music on hold media files, we can simply add the directive called `random`, and set it to `yes`.

A typical Music on Hold class would look something like this:

```
[default]
mode=files
directory=/var/lib/asterisk/music-on-hold
random=yes
```

That's all there is to it! We have now defined two different classes of music. Now, we need to put in an entry for each different stream of MP3s we want our phone system to have.

Queues

As we discussed in Chapter 2, queues give us a logical place to put callers until we are ready to answer the calls. Queues are a very flexible and powerful tool to improve customer service, and better utilize our personnel.

When we edit `/etc/asterisk/queues.conf`, the first sections we will have are `[general]` and `[default]`. Neither of them is used yet.

Below these headings, we come to where we will define our queues. For simplicity's sake, I recommend we name the queue according to the main extension that will represent it. For instance, if on our worksheet we entered that the queue would be dialed from extension 1000, I would define the queue thus:

```
[1000]
```

Now we need to set the parameters for the queue. First, we set the music that the callers will hear until their call is answered. This is done by setting the `music` variable, like this:

```
music = q1000
```

We are stating that the queue should play the music indicated in the `q1000` class of MOH, as defined in `musiconhold.conf`. Each queue can have its own music, or it can use the default MOH.

Next, we define an announcement, if we wish to have one. This will play a audio file to the agent when he or she answers the queued call. We might find this helpful if we have agents answering multiple queues, to tell them which queue calls come from as they answer. If we use this, be sure that we have a file in `/var/lib/asterisk/sounds/<promptfilename>.<extension>` that works.

```
announce = <promptfilename>
```

Notice that we do not need to use the extension in the configuration file: Asterisk will recognize valid files. However, even though we don't have to put the extension on the file name in the configuration file, it must still have the correct extension in the `/var/lib/asterisk/sounds` directory.

Now we need to define our strategy. We should have decided this on our worksheet. For example, if we wanted to use the `ringall` strategy, we would enter:

```
strategy = ringall
```

If we do not define a strategy, Asterisk will default to `ringall`.

We may now define an escape context. We have not yet explored the full meaning of a context, but will be doing so soon. If we checked the box indicating that our users should be able to escape, we will call the context by the name of this queue, followed by the word `out`. Therefore, for this example, it would be:

```
context = 1000out
```

Now we get to set the timeout. This is how long each handset will ring before the queue will consider the call unanswered. We should set this to be the longest reasonable time that we expect our agents to take to pick up the phone. In our case, our agents do not use headsets, and answer questions on the computer. Therefore, they should be given 15 seconds to answer the phone before we assume that they are away from their desk.

```
timeout = 15
```

Next, we define the amount of time we wait before trying all of the extensions again. As our company handles many calls, and we want our customer to have the shortest possible wait time, we will define it to be 0 seconds.

```
retry = 0
```

Now we can set a limit on how many calls will be enqueued. Since we want to allow every caller to enter the queue, we will set this to 0.

```
maxlen = 0
```

On our worksheet, we decided if we wanted to announce callers' positions in the queue, along with an estimate of how long they can expect to be kept in the queue. First, we enter how often we want these announcements. Since I want it to happen every two minutes, I will set it to 120 seconds. If I wanted to have no announcements, I would enter 0.

```
announce-frequency = 120
```

Now we set whether or not to give estimates of waiting time when the current queue position is announced. Valid options are `yes`, `no`, and `once`, meaning the queue application will announce the position only once, when the caller first enters the queue. Since I do want to announce their estimated wait time, I enter:

```
announce-holdtime = yes
```

Next, we can define filenames for the announcement. As the defaults work for us, we will not change them. If we started serving people who spoke a different language, we could change the recordings to use each user's native tongue.

Finally, we define our members. We have a bit of flexibility in defining this. We can either define them as being agents in the queue, or we can hard-code the members to be handsets. Since we want to do things right, we will define the members to be agents.

```
member => Agent/007
member => Agent/777
```

Notice that this only means that we have two members assigned to the queue, Agent 007 and Agent 777, as defined in `agents.conf`.

We can now go on and define other queues, in exactly the same way as above. There is no built-in limit to the number of queues we can define. It depends only on the resources of the server.

Conference Rooms

We now configure our conference rooms. These are defined and configured using the file `/etc/asterisk/meetme.conf`.

This file is among the simplest configuration files we will encounter. When we open the file we see something like:

```
;
; Configuration file for MeetMe simple conference rooms
; for Asterisk of course.
;
[rooms]
;
; Usage is conf => confno[,pin]
;conf => 1234
;conf => 2345,9938
```

As you can see, we have only to define the conference number. We have the option of creating a PIN, to give some level of security to the conference room. In the above configuration file, two conference rooms were created. The first had no password set, and is conference number 1234. The second, conference 2345, will require that people enter the password of 9938 before they can join the conference.

For ease of administration, I suggest we start off using the same conference number as the extension that we're going to call it. As we move around the configuration files we see the advantage of using an extension number as name, especially when trying to troubleshoot a configuration.

Summary

This chapter has walked through the process of configuring the key parts of Asterisk needed by most PBX setups. We looked at settings for:

- Zapata (`zapata.conf`) and Zaptel (`zaptel.conf`)
- Session Initiation Protocol, SIP (`sip.conf`)
- Asterisk's homebrew protocol, Inter-Asterisk eXchange, IAX (`iax.conf`)
- Voicemail (`voicemail.conf`)
- Music on hold (`musiconhold.conf`)
- Call queues (`queues.conf`)
- Conference rooms (`meetme.conf`)

The next step, as we discover in the next chapter, is to create a **dialplan** to tell Asterisk how to handle any particular incoming call.

PACKT
PUBLISHING

[PACKT PUBLISHING]

5
Creating a Dialplan

Congratulations! We have now installed Asterisk, and configured most of your new open-source phone system. Some readers may now be asking, "Are we there yet?" Well, quit whining—it's not far now and we have already come a good way. Let's pause for a moment and review the plan that we created in Chapter 2, where we should have decided what extensions and services we'll provide. Relax, the hard part is behind us, and now we might even start having fun!

In this chapter, we are going to create a full dialplan together. What is a dialplan? Asterisk is a powerful and flexible solution for many different telephony needs. Did you ever wonder how we use all of these features? This is how. At this point, programming experience may well give you an advantage, but it is by no means essential.

When calls come in to the switch, we tell Asterisk step-by-step how to handle the call. Steps can be as simple as playing a sound file to running a customized script. We are limited mostly by our imaginations at this point!

We define all of the steps we want Asterisk to perform in our `extensions.conf` file, in the customary location of `/etc/asterisk`.

Before we begin, we need to set `priorityjumping=yes` in the `[general]` section of `extensions.conf`. This will allow the tips and tricks in this chapter to work with Asterisk 1.2.X.

Creating a Context

What is a context? Simply said, a context is a group of extensions. Each extension must exist within a context. There is more to contexts than grouping extensions though.

In our `extensions.conf` file (or any included files), a context is denoted by square brackets "[]", like so:

```
[mycontext]
. . .
```

So, if a context is a group of extensions, why do we need more than one? Let's think for a minute. Not all employees should be able to dial every phone. Would you trust your 16-year old intern with the ability to dial international calls? I wouldn't. Also, do you want your president to be bothered by customers in the waiting room who use a courtesy phone and misdial? We could find that hazardous to our continued employment!

So, contexts allow us to hide or make inaccessible certain extensions from other extensions. This gives us some level of security. It also allows us to host multiple phone systems on a single server.

Imagine you have two businesses on the same phone system, each with only two handsets. It'd be a pain to have each dial four digits to reach the other handset. We can use contexts to treat each company as if it were on a separate server.

Something very important about contexts is that we can include other contexts through the use of the `include` directive. This means that all extensions in an included context are available. The value of this may not be immediately apparent, but soon we will see the full power of this tool.

Suppose we have some context named `bob`. If we wanted `bob` to include `default`, then we would have the following in our `extensions.conf`:

```
[bob]
include => default
```

This single line placed in any context gives that context the ability to dial any extension in the default context, as well as all contexts included in the default context. This means that if the default context included the `foo` context, then anybody in the `bob` context could dial extensions in the `foo` context.

Suppose we had the following in our `extensions.conf` file:

```
[foo]
exten => 1,1,Playback(tt-monkeys)
include => bar

[bar]
exten => 1,1,Playback(tt-weasels)
```

Now I know that we haven't yet discussed the definition of extensions yet. That's OK. All we need to know is that extension `1` in `foo` will play back a file that sounds like monkeys, and extension `1` in `bar` will play back a file that says that weasels have taken over our phone system.

If we are in context `foo` and press `1`, which file will we hear play? This shows us the danger of includes. We should be careful to not include multiple matches for the same extension. If we do include multiple contexts, the first included context with a match will win. Consider the following file:

```
[foobar1]
include => foo
include => bar
```

```
[foobar2]
include => bar
include => foo
```

If we are in context `foobar1` and press `1`, we will hear monkeys, while if we are in context `foobar2` and press `1`, we will hear weasels. While this can trip the unwary, we will use it to our advantage later on.

Creating an Extension

We all have a good idea of what an extension is. On our legacy PBX, each handset was an extension. Pretty simple, right?

While conceptually simple, there is a little wrinkle. If all we want to do is provide a few handsets, then there's one extension per phone. But Asterisk can do so much more! We need to think of an extension as a group of commands that tells Asterisk to do some things. As amorphous as that may be, it's true.

An extension can be tied to one handset, a queue, groups of handsets, or voicemail. An extension can be attributed to many different areas of the system. If you're familiar with programming terms, perhaps you could say that extensions are polymorphic.

To go further, extensions can be used to provide access to other applications, sound files, or other services of Asterisk. Extensions are important to the magic of Asterisk.

Now that we know why we create extensions let's think about how we create them. Again, they are in the `extensions.conf` file, or any files that you include from there.

> We may decide to break up files such as `extensions.conf` into multiple configuration files. A common example of this is when we create large groups of extensions and choose to give each its own file. This also applies to the other configuration files we use.

The general format for a line in the `extensions.conf` file is:

```
exten => extensionnum,priority,action
```

Let's take a closer look. Each line begins with the command `exten`. This is a directive inside Asterisk. You do *not* change this for each extension.

Next, we have the extension number. Each extension has a unique number. This number is how Asterisk knows which set of commands to run. This extension can be detected in three major ways. First, the phone company may send it in with the calls, as is the case with DID numbers. Users can enter an extension using their touch-tone keys. Finally, there are a few special extensions defined. Some of these are:

- `s`: start extension. If no other extension number is entered, then this is the extension to execute.
- `t`: timeout extension. If a user is required to give input, but does not do so quickly enough, this is the extension that will be executed.
- `i`: invalid extension. If a user enters an extension that is not valid, this is the extension that will be executed.
- `fax`: fax calls. If Asterisk detects a fax, the call will be rerouted to this extension.

Then we have the priority. Asterisk will start at priority 1 by default, complete the requested command, and then proceed to priority *n*+1. Some commands can force Asterisk to jump to priority *n*+101, allowing us to route based on decisions, such as if the phone is busy.

Finally, we have the action. This is where we tell Asterisk what we want to do. Some of the more common actions we may want to perform are:

- `Answer`: This accepts the call. Many applications require that the call be answered before they can run as expected.
- `Playback(filename)`: This command plays a file in `.wav` or `.gsm` format. It is important to note that the call must be answered before playing.
- `Background(filename)`: This command is like `Playback`, except that it listens for input from the user. It too requires that the call be answered first.
- `Goto(context,extension,priority)`: Here, we send the call to the specified context, extension, and priority. While useful, this can be bad style, as it can be very confusing to us if something goes wrong. However, it can be good style if it keeps us from duplicating extension definitions, as moves, adds, or changes would only have to be updated in one place.
- `Queue(queuename|options)`: This command does what it seems like it should. It places the current call in the queue, which we should have already defined in the `queues.conf` file.
- `Voicemail(extension)`: This transfers the current call to the voicemail application. There are some special options as well. If we precede the extension with the letter `s`, it skips the greeting. When we place a `u` before the extension, it uses the unavailable greeting, and a `b` uses the busy greeting.
- `VoicemailMain`: This application allows users to listen to their messages, and also record their greetings and name, and set other configuration options.
- `Dial(technology/id,options,timeout)`: This is where we tell Asterisk to make the phone ring, and when the line is answered, to bridge the call. Common options include:
    - `t`: allow the called user to transfer the call by pressing the # key.

- `T`: allow the calling user to transfer the call by pressing the # key.
- `r`: indicate ringing to the calling party.
- `m`: provide music on hold to the calling party.
- `H`: allow the calling party to hang up by pressing the * key.
- `g`: go on in the context if the destination hangs up.

While this list is not exhaustive, it should be enough to get us started. Suppose we just want to make a Zaptel phone ring, which is on interface 1, and we are going to work completely in the default context. Our `extensions.conf` file would look like:

```
[default]
exten => s,1,Dial(Zap/1)
```

Pretty simple, right? Now, imagine we want to transfer to the voicemail of user 100 if someone is on the phone. Since `Dial` sends you to priority *n*+101 when the line is busy or not available, all we have to do is define what we want to do. Our dialplan would look like:

```
[default]
exten => s,1,Dial(Zap/1)
exten => s,102,Voicemail(b100)
```

Great! We have some of the functionality that users have come to expect. But are you happy yet? The problem is that a phone could ring for years before someone picks it up!

So, for our next exercise, suppose we want to transfer the call to voicemail when the phone is not answered in 30 seconds. So, obviously, we're going to have to use the option in `Dial` to define a timeout. Our dialplan would have something like:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
```

All we're doing is telling Asterisk how to handle the call, in a step-by-step way. It is important to think about all scenarios that a call can go through and plan for them. Just to reiterate a point I made earlier, planning ahead will save us hours of debugging later.

Suppose we want to send anyone who is in a place where they shouldn't be to user 0's voicemail, which will be checked periodically by the receptionist.

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
exten => t,1,Voicemail(s0)
```

All right, we're getting somewhere now! At least we know each call will be handled in some way. What about faxes? Suppose we have only one fax machine (or a centralized fax server) on Zaptel interface 2. Maybe our dialplan should look similar to:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
```

```
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
exten => t,1,Voicemail(s0)
exten => fax,1,Dial(Zap/2)
```

Congratulations! We now have a working phone system. Maybe not the most interesting yet, but we're making great progress. Don't worry, our phone system will grow in features.

Now, to create a list of useful extensions, we need to define a set of commands for each handset we have. Suppose we have three SIP phone users: 1001-1003, with extensions 1001-1003. Our default context would look like:

```
[default]
exten => 1001,1,Dial(SIP/1001|30)
exten => 1001,2,Voicemail(u1001)
exten => 1001,102,Voicemail(b1001)

exten => 1002,1,Dial(SIP/1002|30)
exten => 1002,2,Voicemail(u1002)
exten => 1002,102,Voicemail(b1002)

exten => 1003,1,Dial(SIP/1003|30)
exten => 1003,2,Voicemail(u1003)
exten => 1003,102,Voicemail(b1003)

exten => i,1,Voicemail(s0)
exten => t,1,Voicemail(s0)
exten => fax,1,Dial(Zap/2)
```

For every extension we add, the length of `extensions.conf` will grow by 4 lines (3 lines of code, and one line of whitespace). This is not very easy to read, and it is very easy to make mistakes. There has to be a better way, right? Of course there is!

We can use macros to define common actions. We will create a special **macro context**. The name of these contexts always starts with `macro-`. Suppose we want to call this one `macro-normal`. We would have:

```
[macro-normal]
exten => s,1,Dial(${ARG2}|30)
exten => s,2,Voicemail(u${ARG1})
exten => s,102,Voicemail(b${ARG1})
```

Now, to create the same 3 extensions, we would have:

```
exten => 1001,1,Macro(normal|1001|SIP/1001)
exten => 1002,1,Macro(normal|1002|SIP/1002)
exten => 1003,1,Macro(normal|1003|SIP/1003)
```

So now, each extension we add only requires one extra line in `extensions.conf`. This is much more efficient and less prone to errors. But what if we knew that any four-digit extension beginning with a 1 would be a normal, SIP extension?

Here it is time for us to discuss Asterisk's powerful pattern-matching capabilities. We can define extensions with certain special wildcards in them, and Asterisk will match any extension that fits the description.

Using the underscore (_) character warns Asterisk that the extension number will include pattern matching. When matching patterns, the `X` character represents any number (0-9), the `Z` character will match the numbers 1-9, the `N` character represents numbers 2-9, and the period (`.`) represents a string of any number of digits.

Also, we can use certain variables in our dialplan. One such variable is `${exten}`, which represents the extension that was used.

So, for this example, we could use the following definition:

```
exten => _1XXX,1,Macro(normal|${exten}|SIP/${exten})
```

This one line of code has now defined 1000 extensions, from 1000 to 1999. All we have to do is ensure that our voicemail user, extension, and SIP user are all the same number. Pretty cool, huh?

One other note: if we wish to modify the behavior of all extensions, all we have to do is modify the macro. This should help us quite a bit as we tweak Asterisk to fit our business needs.

Creating Outgoing Extensions

With this dialplan, we have only catered for incoming calls. Of course we will want to create extensions to dial out.

How these outgoing extensions look depends on the plan we made earlier. It also depends on how you want the switch to act for your users. We always want to make it as similar to any old system as possible to reduce the need to retrain users.

Most phone systems require a user to dial a certain digit to designate the call as one that has a destination outside the switch. In our legacy PBX, we accessed outgoing lines by dialing a 9. To copy this behavior, we could do something like:

```
[outgoing]
exten => _9.,1,Dial(Zap/g1/${EXTEN:1})
```

Notice that we are using pattern matching as we did before. Notice also that we used `g1` as the Zaptel interface. This is a good use for a group. This simply means "any free interface in group 1". Thus, if we put all of our outgoing lines in group 1, then when we dial an outside number, we do not have to guess which channel is free.

Remember that the variable `exten` represents the extension that we are in. If a user dials 95555555, then `exten` is equal to 95555555. By using `${EXTEN:1}`, we instruct Asterisk to strip the first (leftmost) digit. Thus, `${EXTEN:1}` equals 5555555. If we wanted to strip the two leftmost digits, it would be `${EXTEN:2}`.

Many discount long distance carriers will charge the same for local calls as long distance. In a case like that, we would want to make sure that local and toll-free calls went out on lines connected to the local telephone company, while calls destined for long-distance

locations should go out through our discount carrier. For this example, we will assume that the discount carrier is Zaptel group 1, and the local telephone company is Zaptel group 2.

```
[outgoing]
exten => _9N.,1,Dial(Zap/g2/${EXTEN:1})
exten => _91.,1,Dial(Zap/g1/${EXTEN:1})
```

Therefore, if the number dialed is preceded with a 9 and a 1, then the call will go out through our Zaptel group 1 lines, and if it does not have a 1, it will go out through our group 2 lines.

What if we had used:

```
[outgoing]
exten => _9X.,1,Dial(Zap/g2/${EXTEN:1})
exten => _91.,1,Dial(Zap/g1/${EXTEN:1})
```

Notice that this is a problem. When we dial 9, followed by 1, we have two statements that match. When you start Asterisk, you will notice that the system parses the `extensions.conf` (and included files) and immediately reorders all of the extensions before building a database file. Therefore, it is ambiguous as to which statement it will execute.

While this method is technically functional, it is not the best way to do it. A better way would be:

```
[outgoing]
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXNXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
```

This does not allow for medium distance dialing. It also does not handle the case of all lines being busy in one of the groups. Let's see how we could take care of that:

```
[outgoing]
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _9NXXXXXX,2,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXNXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
```

This is getting much better, but what about when all of our lines are busy, for both groups? We should probably notify the user that their call didn't go through because all lines are in use. We could do something like this:

```
[outgoing]
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _9NXXXXXX,2,Dial(Zap/g1/${EXTEN:1})
exten => _9NXXXXXX,3,Congestion
exten => _91NXXNXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXNXXXXXX,3,Congestion
exten => _91NXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXXXXX,3,Congestion
```

Imagine a phone switch in which all extensions were exactly four digits long. How would this knowledge affect our outgoing lines? Perhaps the fundamental question is whether we really need to dial the 9 at all. While people generally expect to dial a 9 at work, they do not seem to care to do so from home. Maybe we should do this:

```
[outgoing]
exten => _NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _NXXXXXX,2,Dial(Zap/g1/${EXTEN:1})
exten => _NXXXXXX,3,Congestion
exten => _1NXXNXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _1NXXNXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _1NXXNXXXXXX,3,Congestion
exten => _1NXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _1NXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _1NXXXXXX,3,Congestion
```

There is a drawback to this configuration. If you are using Zaptel interfaces, and people are touch-tone dialing, Asterisk will have to wait for a timeout period to complete dialing some four-digit extensions. The reason for this is simple. Any extension number that partially matches the beginning of the patterns above will have to be held to see if more digits are coming. However, SIP and IAX phones do not seem to suffer the same issue.

Because of this issue, from here forward we will assume that all outgoing calls are preceded with the digit 9.

We have already discussed that contexts can provide security for outgoing phone calls, but this example fails to describe such security. Suppose you have two groups of employees, those who may make toll calls, and those who may not. All employees have an individual handset. The most logical choice is to make two contexts:

```
[longdistance]
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _9NXXXXXX,2,Dial(Zap/g1/${EXTEN:1})
exten => _9NXXXXXX,3,Congestion
exten => _91NXXNXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXNXXXXXX,3,Congestion
exten => _91NXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXXXXX,3,Congestion

[local]
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _9NXXXXXX,2,Dial(Zap/g1/${EXTEN:1})
exten => _9NXXXXXX,3,Congestion
```

We simply place each handset into one of the two contexts, based upon what numbers we want them to be able to dial. While this would work, we duplicated three lines between the contexts. Remember when we discussed contexts, and I mentioned that we can use the `include` directive? Here's a good place to do so. So now, we have:

```
[longdistance]
exten => _91NXXNXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
```

```
exten => _91NXXNXXXXXX,3,Congestion
exten => _91NXXXXXX,1,Dial(Zap/g1/${EXTEN:1})
exten => _91NXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXXXXX,3,Congestion
include => local
[local]
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _9NXXXXXX,2,Dial(Zap/g1/${EXTEN:1})
exten => _9NXXXXXX,3,Congestion
```

Let's take a quick look at what we have. We can now place handsets in the `local` or `longdistance` context. This gives us a good bit of security, but there is one small problem. Do you see it? If we put our users into either context, they cannot dial internal extensions. Assuming we allowed calling our local extensions in the default context, we should update the `local` context like so:

```
[local]
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _9NXXXXXX,2,Dial(Zap/g1/${EXTEN:1})
exten => _9NXXXXXX,3,Congestion
include => default
```

By so doing, we give access to the extensions for the `local` context. We also give access to the default context for those in the `longdistance` context because `longdistance` includes `local`, which in turn includes `default`.

Thus, we must be very careful about what we include. When we include any context, we are in turn including all contexts it includes. It is easy to include our way out of the security we set up!

Advanced Call Distribution

What exactly is Advanced Call Distribution? Many phone systems tout this feature, but most do not adequately define what it means. Basically, it refers to using call queues, parking calls for another user to answer, and Direct Inward Dialing (DID).

So that we keep our focus, we will look at each of these elements individually.

Call Queues

In Chapter 4, we configured call queues, through the `/etc/asterisk/queues.conf` file. As we go through how we're going to use our queues, we may decide we want to change the way our queues are configured. There is absolutely no problem with changing the configuration so that it more accurately reflects our needs. Just remember that we need to issue a reload on the Asterisk console, or type `#asterisk –r –x reload` at the command line.

The power and flexibility of other ACD systems can be matched or exceeded by Asterisk. As we evaluate our needs, we should remember that configuring a single aspect of Asterisk sometimes requires changes to more than one file. For example, queues will be

configured both in the `queues.conf` file and the `extensions.conf` file. Since we have already discussed the `queues.conf` file in Chapter 4, we will discuss how to set up `extensions.conf` to give us the desired result.

When dealing with call queues, we need to think about the two types of users we have. First, we have the caller who calls in and waits in the queue for the next agent. We can think of this person as our customer. Next, we have the agents who work the queue. We can think of these people as our users.

As a business, we have to decide what we want our customers' experiences to be. Our call queue can make it sound like a phone is ringing. Or we can use music on hold while the customer waits. We can also announce call position and estimated wait time if we want to.

When we place customers in a queue, we use the `Queue` application. To place a caller in the queue named `bob`, we would use something like:

```
exten => 1000,1,Queue(bob)
```

Suppose we have an operator's extension. Since Ollie the Operator may have more than one call at a time, we decide to give him a call queue. His calls are always about a minute long. The customers waiting for him are going to there because they got lost in a system of menus. His queue will be named `operator`.

In this instance, we will choose to have the customer hear ringing so they will believe they are about to be helped. The sound of ringing should not last more than about a minute. We will not announce call queue length because our customer should not know that he or she is in a queue.

The entry for this queue would be:

```
exten => 0,1,Queue(operator|tr)
```

Notice our use of options. Options for the queue application include

- `t`: Allow the user to transfer the customer.
- `T`: Allow the customer to transfer the user.
- `d`: This is a data-quality call.
- `H`: Allow the customer to hang up by hitting `*`.
- `n`: Do not retry on timeout. The next step in the dialplan will be executed.
- `r`: Give the customer the ringing sound instead of Music on Hold.

Thus, we told the `Queue` application to give the customer ringing, and the user (Ollie) the ability to transfer calls (since he's the operator).

Now, suppose we have Rebecca, the receptionist. Rebecca is at SIP phone 1006. When Ollie goes to the bathroom, we want our poor lost customers to be routed to her. So we could use the following in our `extensions.conf` file:

```
exten => 0,1,Queue(operator|trn)
exten => 0,2,Dial(SIP/1006)
```

Now Rebecca had better answer this. Until she does, the phone will continue to ring. Notice that this call will never end up in Rebecca's voicemail, as it is not transferred to her extension, but instead dials her phone directly.

We have adequately addressed the customer's experience. But now we need to look at how our users will join and leave the queue. Previously, we discussed the power and flexibility of using Agents in queues. As with most things in Asterisk, there are many ways we can associate members to queues. The three main ways are statically, dynamically, and using agents.

Our first option is to have members statically assigned to the queue. To do this, we use the `member` directive in the `queues.conf` file. This is most helpful when we have a queue with fixed members, such as a switchboard queue.

Our second option is to allow members to log in dynamically. We do this through the `AddQueueMember` application. An example of this would be:

```
exten => 8101,1,AddQueueMember(myqueue|SIP/1001)
```

Whenever anybody dials extension 8101, the telephone handset SIP/1001 would be added to the queue named `myqueue`. All that we would have to do is define a login extension for every member of every queue.

What happens when this member no longer wishes to be in the queue? We use the `RemoveQueueMember` application, like this:

```
exten => 8201,1,RemoveQueueMember(myqueue|SIP/1001)
```

With this configuration, whenever anybody dials extension 8201, the telephone handset at SIP/1001 is removed. Again, we would have to define a logout extension for each member of the queue.

Suppose we did not wish to have to define a login and logout extension for each member. We have the option of leaving off the interface (`SIP/1001` in the above example) and having Asterisk use our current extension. While this is very useful, Asterisk does not always use the right value. However, if it works for all extensions that need to be in the queue, we would only have to define one login and one logout per queue. The code would look like:

```
exten => 8101,1,AddQueueMember(myqueue)
exten => 8201,1,RemoveQueueMember(myqueue)
```

This is better than having to define a login and logout for each member of each queue, but sometimes users are not good about remembering multiple extensions to dial. The `AddQueueMember` application will jump to priority *n*+101 if that interface is already a member of the queue. Therefore, we could define an extension like:

```
exten => 8101,1,Answer
exten => 8101,2,AddQueueMember(myqueue)
exten => 8101,3,Playback(agent-loginok)
exten => 8101,4,Hangup
exten => 8101,103,RemoveQueueMember(myqueue)
```

```
exten => 8101,102,Playback(agent-loggedoff)
exten => 8101,105,Hangup
```

When we define it this way, a user dialing extension 8101 is logged in if not already a member of the queue, or logged out if in the queue. Also, we added a confirmation to the action, so that the user can know if they are now in or out of the queue. Notice that before we could use the `Playback` application, we had to answer the call. If we have a lot of these, we could define a macro extension, like:

```
[macro-queueloginout]
exten => s,1,Answer
exten => s,2,AddQueueMember(${ARG1})
exten => s,3,Playback(agent-loginok)
exten => s,4,Hangup
exten => s,103,RemoveQueueMember(${ARG1})
exten => s,104,Playback(agent-loggedoff)
exten => s,105,Hangup
. . .
[default]
exten => 8101,1,Macro(queueloginout|queue1)
exten => 8102,1,Macro(queueloginout|queue2)
exten => 8103,1,Macro(queueloginout|queue3)
```

And thus we see that using a macro will save us five lines in our `extensions.conf` for every queue after the first. This is how we can add queue members dynamically.

Our final option for adding queue members is by using Asterisk's agent settings. We were able to define agents in `/etc/asterisk/agents.conf`. We create an agent by defining an ID and a password, and listing the agent's name.

In the `queues.conf`, we could define agents as members of queues. Calls will not be sent to agents unless they are logged in. In this way, queues can be both dynamic and static: they are static in that we do not change the members of the queues, but dynamic in that calls will go to different handsets based upon which agents are logged in.

There are two main types of agents in this world. There are the archetypical large call center agents who work with a headset and never hear ringing, and there are the lower-volume agents whose phone rings each time a call comes in. Asterisk has the flexibility to handle both types of agents, even in the same queue.

First, imagine a huge call center that takes millions of phone calls per day. Each agent is in multiple queues, and we have set each queue to use an announcement at the beginning of calls to let the agent know which queue the call came in from. As employees arrive for their shift, they sit down at an empty station, plug in their headset, and log in. Each employee will hear music in between calls, and then hear a beep, and the call will be connected. To accomplish this, we use the line:

```
exten => 8001,1,AgentLogin
```

Through the normal login, the call is kept active the whole time. The agents will logout by hanging up the phone. This allows large call centers to be quieter, as the distraction of ringing phones will be removed. It also allows for more efficient answering of lines, as the time required to pick up the phone is eliminated.

If we have employees who have other work to do as well as answer calls, or have to call other employees to consult, or simply have a low call volume, we may wish to use `AgentCallbackLogin`. When we do so, our agents' phones are rung when a call comes in. We do this by using:

```
exten => 8001,1,AgentCallbackLogin
```

When our users arrive at work and wish to login, they call extension `8001`, where they are prompted for their agent ID, password, and then an extension number at which they will take calls. This is how Asterisk knows how to reach them. Our agents can log out when using `AgentCallbackLogin` by going through the same procedure as for login, except when they are prompted for their extension, they press the # key.

It may be a good idea for us to review `agents.conf`. If we defined `autologoff`, then after the specified number of seconds ringing, the agent will be automatically logged off. If we set `ackcall` to `yes`, then agents must press the # key to accept calls. If we created a `wrapuptime` (defined in milliseconds), the Asterisk will wait that many milliseconds before sending another call to the agent. These options can help us make our phone system as user-friendly as we want it to be.

Through the use of call queues, we can distribute our incoming calls efficiently and effectively. We have plenty of options, and can mix-and-match these three ways of joining users to queues.

Call Parking

In many businesses across the United States, an operator can be heard announcing "John, you have a call on line 3. John, line 3." In Asterisk, we don't really have lines in the way that analog PBXs have them. Our users are accustomed to not having to transfer calls, especially when they may not know exactly where John is.

Asterisk uses a feature known as Call Parking to accomplish this same goal. Our users will transfer calls to a special extension, which will then tell them what extension to call to retrieve the call. Then our users can direct the intended recipient to dial that extension and connect to the call.

In order to be able to use this feature, we must define our parking lot. This is done in the `/etc/asterisk/parking.conf` file. In this file, there are only have a few options that we will need to configure. First, we must create the extension people are to dial to park calls. This can be whatever extension is convenient for us. Then we will define a list of extensions on which to place parked calls. These extensions will be what users dial to retrieve a parked call. Next, we will define what context we want our parked calls to be in. Finally, we will define how many seconds a call remains parked before ringing back to the user who parked it. Here is an example:

```
[general]
parkext => 8100
parkpos => 8101-8199
context => parkedcalls
parkingtime => 120
```

These settings would mean that we can park calls by dialing 8100, and the call will be placed in extensions 8101 through 8199, giving us the ability to have up to 99 parked calls at any given time. The calls will be in the context called `parkedcalls`, which means we should be careful to include it in any context where users should be able to park and retrieve calls.

When our users transfer a call to extension 8100, they will hear Asterisk read out the extension that the call has been placed on. They can now make a note of it and notify the appropriate coworker of the extension to reach the calling customer on. If the call is not picked up within the given `parkingtime`, then the call will ring back to the user who parked the call.

By using call parking, we can help our users by providing a feature similar to that of previous generations of PBXs. This also allows users to collaborate and redirect callers to other users who are better equipped to handle our customers' needs.

Direct Inward Dialing (DID)

Suppose we work at a healthcare company with over 100 employees. We have two PRI lines coming in, and only three switchboard agents to handle incoming calls. As a healthcare company, we schedule many appointments, answer questions about prescriptions, and help patients with billing questions. These three agents are always busy.

Now suppose the IT guy's wife calls in to ask if he wants sprouts or mash with his dinner. Do we want our switchboard agents to have to answer the call, find out who it is and what they want, and then transfer the call, or would we rather the IT guy's wife can call her husband directly?

This is where Direct Inward Dialing comes in handy. DID is a service provided by phone companies where they send an agreed-upon set of digits, depending on the number the customer dialed. For most phone companies, the sent digits will be the full 10-digit number (in the United States). But this can be as small as the last digit.

All right, so the phone company is sending digits. What are we going to do with them? Imagine you have a PRI coming in to your office, and only ten phone numbers, a block from (850) 555-5550 to 5559. Your phone company has agreed to send you only the last digit dialed, which will be from 0 to 9, because you are guaranteed for this to be unique. Asterisk can route calls based on this DID information.

If we have our PRI line's channels defined to go into a context called `incoming`, this context could look like:

```
[incoming]
s,1,Goto(default,s,1)
i,1,Goto(default,s,1)
t,1,Goto(default,s,1)

0,1,Goto(default,1234,1)
1,1,Goto(default,2345,1)
2,1,Goto(default,3456,1)
3,1,Goto(default,4567,1)
4,1,Goto(default,5678,1)
5,1,Goto(default,6789,1)
6,1,Goto(default,7890,1)
7,1,Goto(default,1111,1)
8,1,Goto(default,1111,1)
9,1,Goto(default,1111,1)
```

There are a few things we should notice about this. First, we handled the error cases. What if a glitch at the phone company results in four digits being sent? We cannot allow a simple mistake on their end to interrupt our ability to receive phone calls.

Secondly, we are using `Goto` statements. We've briefly discussed how they can be both good and bad. In this case, by using a `Goto`, if a user moves from one extension to another, we do not have to update it everywhere, only in the default context.

Finally, we are allowed to send multiple incoming DIDs to the same extension, if we so desire, as in the last three lines shown above. This might be useful if extension 1111 is the operator, and we do not yet have the number 7, 8, or 9 assigned to a user.

Of course, in real life this is going to get much more complicated, as phone numbers will probably come in with the full 10 digits. But the concept is the same: we can define extensions based upon information that the phone company sends when the call is established.

By using DIDs, we can cut down on bottlenecks and give direct access to certain extensions. This tool of Asterisk helps make our phone system fast, efficient, and friendly to our users and customers.

Automated Attendants

Any time we call a large company, we are greeted by a computer voice, asking us to route our call based on what we want or need. We are all familiar with call menus. While we won't get into a philosophical debate about how good or bad they are, we will talk about how to make them.

Suppose we want to create a menu of options, such as, "For a billing question, press 1, to request a configuration change press 2 …." Now suppose you press 1, and you hear the option of "For help reading your statement, press 1, if you wish to dispute a charge, press 2, …" This is just a standard phone tree, with which most users are comfortable. Asterisk knows which extension to execute based upon what context we are currently in.

Suppose that your customer service reps are on SIP/1000, and the manager whom you wish to handle all disputes is on SIP/1001. Then, you have techs on SIP/1002 and SIP/1003. Our configuration file could look like:

```
[mainmenu]
exten => s,1,Answer
exten => s,2,DigitTimeout(5)
exten => s,3,ResponseTimeout(30)
exten => s,4,Background(welcome)
exten => s,5,Background(options)

exten => 1,1,Goto(billing,s,1)
exten => 2,1,Dial(SIP/1002&SIP/1003)

[billing]
exten => s,1,Answer
exten => s,2,DigitTimeout(5)
exten => s,3,ResponseTimeout(30)
exten => s,4,Background(billingoptions)

exten => 1,1,Dial(SIP/1000)
exten => 2,1,Dial(SIP/1001)
```

There are some very important points to make here. It is good to define your digit and response timeouts each time you are going to give an option to the user. This makes your dialplan easier to understand and maintain. Also, notice that we must use the `Answer` command before being able to playback files. Remember that `Background` allows for user input to be captured during the file, while `Playback` does not.

Also notice that we issued the `Answer` command in the billing menu as well. If we know that nobody could ever get into the billing menu without having passed through the main menu, we could probably leave the command out. However, it does not have any serious adverse reactions, and by leaving it in, we are able to offer a direct line to the billing department, if we ever choose to do so.

Suppose we want to have users be able to dial any extension at any time. All we have to do is use the magical `include => default` in each of the contexts. This can cause the same delay in dialing we discussed previously in the section *Creating Outgoing Extensions*, so it should be used with care. Also, users sometimes mash buttons at random, and may stumble across random extensions, frustrating the user. Some organizations have chosen to have users press a specific digit to be able to dial extensions directly to deal with these problems.

On another design note, most customers do not like being trapped for long periods of time in menu systems. Care should be taken to ensure menus do not get too deep. Also, it is easier for customers if you don't give too many options in one level.

Care must also be taken in selecting invalid and timeout responses. But most importantly, we have to do something. Leaving a customer in limbo with nowhere to go and no prompts to get them there is not a friendly move.

Let's build on our previous system with these concepts in mind. Of course, the prompts would be updated to suggest using the new features. For the purposes of this example, let's assume our receptionist is on SIP phone 1004.

```
[mainmenu]
exten => s,1,Answer
exten => s,2,DigitTimeout(5)
exten => s,3,ResponseTimeout(30)
exten => s,4,Background(welcome)
exten => s,5,Background(options)

exten => 1,1,Goto(billing,s,1)
exten => 2,1,Dial(SIP/1002&SIP/1003)
exten => 3,1,Goto(dialbyext,s,1)

exten => t,1,Goto(s,1,1)
exten => i,1,Goto(s,1,1)

exten => 0,1,Dial(SIP/1004)

[billing]
exten => s,1,Answer
exten => s,2,DigitTimeout(5)
exten => s,3,ResponseTimeout(30)
exten => s,4,Background(billingoptions)

exten => t,1,Goto(billing,s,1)
exten => i,1,Goto(billing,s,1)

exten => 1,1,Dial(SIP/1000)
exten => 2,1,Dial(SIP/1001)
exten => *,1,Goto(mainmenu,s,1) ; escape to the previous menu

exten => 0,1,Dial(SIP/1004)

[dialbyext]
exten => s,1,Answer
exten => s,2,DigitTimeout(5)
exten => s,3,ResponseTimeout(30)
exten => s,4,Backgroud(enterextension)

exten => i,1,Playback(invalid)
exten => i,2,Goto(dialbyext,s,1)

exten => t,1,Playback(imsorryididntgetthat)
exten => t,2,Goto(dialbyext,s,1)

exten => *,1,Goto(mainmenu,s,1)

exten => 0,1,Dial(SIP/1004)

include => default
```

This is a pretty good system. We can build whatever we need using these concepts. Each menu system will be different, based on your needs. As your needs change, the menu system is very easy to update. As a reminder, when you have updated the dialplan, you can refresh it by executing the `reload` command at the console. This command does not interrupt calls that are currently in progress.

We should take a moment and discuss what a good menu is, and what a bad menu is. We should put our most commonly chosen option first. We should not confuse our customers with too many choices. We should have errors that take you back one step, instead of all the way to the beginning of the menu. We should give an operator who can help people if they get lost. Finally, we should strive to not have our menu any deeper than four steps.

If we keep these design principles in mind, our Automated Attendant can be the best employee we've ever had. No benefits, never sick, and always cheerful and ready to help!

System Services

We have talked about how to create contexts, extensions, and how to make our system powerful. Some of the ideas we have discussed will be useful in some situations, yet not applicable in others. All of them work together to make Asterisk a flexible solution for many different needs. There are still other uses that we won't get into, as the need for them is less frequent.

There are some basic system services that we have not yet discussed. In Chapter 4, we configured our voicemail users. Asterisk also includes an application called Directory that reads the voicemail configuration and allows callers to look up an extension based on the user's last name. In the section *Creating an Extension* earlier in this chapter, we saw how to send calls to voicemail. How do users retrieve messages? How do we use this Directory?

```
exten => 8000,1,VoicemailMain(@default)
exten => 8888,1,Directory(default)
exten => 8888,2,Goto(1)
```

As you can see, providing access to the Directory (extension 8888) and voicemail (extension 8000) is easy. But priority 2 of extension 8888 may seem odd. This is in here because the Directory seems to crash from time to time. By setting priority 2 to place the user back into the Directory, the failure is almost transparent to our users.

In priority 1 of extension 8000, we tell Asterisk to send people to voicemail, in the default context. If we have only one context, we can usually get away without defining the context; however, it is better to be safe. In the Directory application, we must define what context to use, which we specified as `default`. Both of these contexts should match the context listed in the `voicemail.conf` file.

When our users call the Directory, they will be prompted to enter the first three letters of the person's last name. Pulling from the `voicemail.conf` file by default, Asterisk will search for all matches. If users have recorded their name (option 0, 3 in `VoicemailMain`), then Asterisk will play the file where the person speaks their name; otherwise, Asterisk will spell the complete name out. When it finds the person our user is looking for, they press 1. Asterisk immediately tries to dial that person.

Here we need to talk about naming our contexts. If our voicemail context is called `foo`, then Asterisk will try to dial `extension@foo`. However, if our extensions are all defined in a context called `bar`, then the Directory will fail. Therefore, we must make sure that the contexts in `voicemail.conf` (where we define the voicemail entry) and the contexts in `extensions.conf` (where we define the extension) match.

Now suppose we want to record prompts for our menus. Asterisk can play standard Windows `.wav` files; however, getting the files recorded and into the phone system may not be the most convenient thing to do. Therefore, we can create a simple extension to allow us to record a prompt. We will allow users to input a four-digit name for the file so that they can record many prompts before having to sort them. The prompts will be stored in `/tmp`, and be recorded as `.wav` files. We will assume we have a file called `enter4digits` and `record-instructions`, as well as a file called `1toaccept2torerecord3torecordanother`.

```
exten => 8200,1,Goto(record,s,1)

[record]
exten => s,1,Answer
exten => s,2,Read(RECORD|enter4digits|4)
exten => s,3,Playback(record-instructions)
exten => s,4,Record(/tmp/recording-${RECORD}|wav)
exten => s,5,Wait(2)
exten => s,6,Playback(/tmp/recording-${RECORD})
exten => s,7,ResponseTimeout(10)
exten => s,8,Background(1toaccept2torerecord3torecordanother)

exten => 1,1,Hangup
exten => 2,1,Goto(s,3)
exten => 3,1,Goto(s,2)
```

This little context will give us the option of recording whatever custom prompt we want, in a `.wav` format. We could add a timeout extension and an invalid extension if we so desire, just as we have done in other contexts.

Another great service that Asterisk provides is conferencing. We configured `meetme.conf` in Chapter 4 so that we could have conference rooms. In order for users to be able to enter the conference rooms, we must create an extension giving us access.

Suppose we have 10 conference rooms, which we want to place on extensions 8900 to 8909. We also named our conferences 8900 to 8909 in `meetme.conf`. Our `extensions.conf` should contain:

```
exten => 8900,1,MeetMe(8900)
exten => 8900,2,Goto(default,s,1)

exten => 8901,1,MeetMe(8901)
exten => 8901,2,Goto(default,s,1)

exten => 8902,1,MeetMe(8902)
exten => 8902,2,Goto(default,s,1)

. . .

exten => 8909,1,MeetMe(8909)
exten => 8909,2,Goto(default,s,1)
```

If we use a little bit of variable magic, we can get these lines down to:

```
exten => _890X,1,MeetMe(${EXTEN})
exten => _890X,2,Goto(default,s,1)
```

And so we see that by making our `MeetMe` conference number the same as the extension number users dial to join, our lives are made a little bit easier. Also, by having all of our conferences in a block of 10 or 100, we are able to use pattern matching to make our `extensions.conf` shorter.

Another service we may wish to have is a `MusicOnHold` extension. It will do nothing but play the music on hold that is currently running. This can be a useful tool for the administrator to check if the music is even running, or to check the volume on a handset. To add a `MusicOnHold` extension, we would add something like:

```
exten => 8010,1,MusicOnHold(default)
```

This extension will play the music that is playing on hold in the default class, as configured in the `musiconhold.conf` file. It will only end when the caller hangs up the phone. But we need to remember that it is not like the background music of some phone systems, in that it does tie up the line that is listening to the music. If calls come in while a user is listening to the music on hold, they will go through the normal procedure for a phone being busy.

Summary

In this chapter, we have looked at how to create a dialplan for our Asterisk system. We looked at how to set up:

- Contexts
- Extensions for incoming calls
- Extensions for outgoing calls
- Call Queues
- Call Parking
- Direct Inward Dialing
- Voicemail
- Automated Phone Directory
- Conference Rooms

Although this list of available services is not exhaustive, it is certainly enough to get our phone system up and running. There are many further options available to us, which we will try out as we work through various case studies later in this book. These case studies will give us the full configuration of some Asterisk installations for fictitious companies. We will be able to view the configuration files and keep those parts that help us meet our particular needs.

PACKT
PUBLISHING

[PACKT PUBLISHING]

6
Quality Assurance

The world has changed quite a bit in the last 150 years. Over this time, the telephone system has been invented, improved, and automated. Telephone switches no longer refer to people sitting in a large room connecting wires between the appropriate jacks. Flexible and powerful telephone service has moved from a dream to an expectation in large businesses, and for most of us it is a necessity.

Today, telephone systems are the lifeblood of business. They are how we take orders, acquire supplies, and even call for emergency assistance. With the increase in prominence of telephones, the expectations of telephone users have increased proportionally.

Not only have the technological expectations for telephone systems increased dramatically, but consumers are expecting more and more out of the businesses they call. Customers expect to be helped quickly and professionally. They want to know everything in a matter of moments. Roads do not hold the only rage our society is facing today. As a business we have a variety of questions relating to our telephone system. How are our personnel handling angry callers? Are our employees answering the calls in a reasonable amount of time? Do we have any workers using the phone system for personal calls when they should be doing their job?

We will never be able to make sure everybody does what they are supposed to do all of the time. What we will be able to do at the end of this chapter is perform spot-checks on how we are doing on customer service, and make sure our phone service isn't being used for unauthorized purposes. Ultimately, it comes down to a matter of trust; however, some people do not know better because they haven't been fully trained. Most will always act honorably; however, some just cannot and should not be trusted. We will try to find out who is who.

Call Detail Records

When we talk about *security*, what images come to mind? Maybe a big, burly guard? Perhaps a bunch of guys in green, carrying machine guns? Do we imagine a person with a metal-detecting wand? Or do we think of thick glass windowpanes?

All of these are security features. It is just that some are a little more intrusive than others. Each time we increase security, we become a little bit less friendly. We all have to decide how far we are willing (and able) to go.

In the continuum of security, Call Detail Records are the least intrusive. No special usernames or passwords have to be remembered. No fear of big brother breathing down your customers' and users' necks need be felt. We are simply doing the same thing telephone companies do: tracking what calls were made, when they were made, how long they lasted, where they came from, and a few other bits of information. This information is then available for us to review at our leisure.

Asterisk gives us a few options on how we track this information. The two major choices are flat-file logging and database logging.

Flat-File CDR Logging

By default, Asterisk includes a module called `cdr_csv`. Right out of the box, Asterisk logs all calls coming in and going out. The information for these calls is placed in a comma-separated value (CSV) file. This CSV file is located in `var/log/asterisk/cdr-csv`. All information is available in `Master.csv`, and some channels can be configured to send some information to other files as well.

The benefit of using a CSV file is the simplicity. Right after compiling and installing Asterisk, this method will work. No additional configuration is required. Also, no additional network traffic is generated, and no additional services have to be installed on our server.

When using the CSV form of CDR, we will see lists and lists of values. They are not very easy to parse, so here is the format, in the order in which they appear:

1. account code: as determined by the channel (for Zap) or the user (for IAX and SIP)
2. source: the source of the call
3. destination: the destination of the call
4. destination context
5. caller ID
6. channel: the channel of the source
7. destination channel: if applicable
8. last application: the last application run on the channel
9. last application argument: the last argument to the last application on the channel
10. start time: the time the call commenced
11. answer time: the time the call was answered

12. end time: the time the call ended
13. duration: the difference between start time and end time
14. billable seconds: the difference between answer time and end time, which MUST be less than the duration
15. disposition: either `ANSWERED`, `NO ANSWER`, or `BUSY`
16. `amaflags`: as set for the channel or user, like account code
17. `uniqueid`: a unique call identifier
18. `userfield`: a user field set by the `SetCDRUserField` command

We see that there are many items of information logged for each and every call. We can compare the billable seconds with our phone bill at the end of the month to make sure they're close. We can look at the destination and figure out if the calls were authorized. This gives us enough information to answer most questions we may have about a phone call.

While we have enough information to answer questions, finding that answer is not very easy. We would have to scan through the entire file to try to find anything. If we are going to use an accounting package or reporting software, CSV may be exactly what we need; however, if we wish to use it in a more ad hoc sort of way, it is not very readable.

Database CDR Logging

If we wish to read our CDR logs, it is most easily accomplished when the records are sortable. The easiest way to do this is to store our CDR records in a database.

Even in this, Asterisk gives us choices. Included with Asterisk is support for PostgreSQL databases. In order to be able to install this, we must first have the `postgresql-devel` package installed on our system. If you have to install this package, you'll need to reinstall Asterisk. The automake system will automatically detect that we have the capability to use PostgreSQL and compile that module for us.

Aside from the development packages we have installed, we will also need a PostgreSQL server somewhere on our network. It can be the same machine as the Asterisk server, but it doesn't necessarily need to be. In fact, it probably makes sense to have only one such database server on our network, and we don't want to tie up too much of our PBX's resources with database maintenance and storage.

There is a script in `/usr/src/asterisk/contrib/scripts/` called `postgres_cdr.sql`, which creates the correct table structure for us. This script should be run from the database server.

If we get an error message while rebuilding that says something like "cannot find `-lz`", then we need to install `zlib-devel`.

Now that we have set up our database and installed the CDR module, we must configure Asterisk to use the correct database. To do this, we need to edit `/etc/asterisk/cdr_pgsql.conf`. All of the configuration variables are in the global section. Our file should look something like:

```
[global]
hostname=dbserver.mydomain.tld
port=5432
dbname=asterisk
password=supersecret
user=asteriskuser
```

Once we have these variables set, the next time we restart Asterisk, all CDR records will be logged in the database.

If PostgreSQL is not our database of choice, we can use MySQL. This is not part of the normal distribution of Asterisk. But since we have already installed `asterisk-addons`, we should already have the ability to use MySQL for CDR logging.

Before we compile, we need to make sure we have `mysql-devel` installed. First, we need to decide which version we're going to use. Because of some license quibbles, MySQL version 4.0 and later is not in the automatic package distribution chain. Instead, if we do need to download it, we will have to get it directly from `www.mysql.com`. However, the older version (3.x) will work with Asterisk so you may wish to take a look at the differences between what version 3 offered and what later versions give us.

Other than the development package mentioned above, we will also need a MySQL server somewhere on our network. Just as with PostgreSQL, we can choose to have it on the same server as Asterisk, but for the same reasons, we probably shouldn't.

Next, on the database server, we need to create the database with a user and a table for the CDR data. We do this by running:

```
# mysqladmin create database asteriskcdrdb
# mysql
mysql> GRANT ALL PRIVILEGES
    -> ON asteriskcdrdb.*
    -> TO asteriskcdruser
    -> IDENTIFIED BY 'changethis2yourpassword';
mysql> USE asteriskcdrdb;
mysql> CREATE TABLE cdr (
    -> uniqueid varchar(32) NOT NULL default '',
    -> userfield varchar(255) NOT NULL default '',
    -> accountcode varchar(20) NOT NULL default '',
    -> src varchar(80) NOT NULL default '',
    -> dst varchar(80) NOT NULL default '',
    -> dcontext varchar(80) NOT NULL default '',
    -> clid varchar(80) NOT NULL default '',
    -> channel varchar(80) NOT NULL default '',
    -> dstchannel varchar(80) NOT NULL default '',
    -> lastapp varchar(80) NOT NULL default '',
    -> lastdata varchar(80) NOT NULL default '',
    -> calldate datetime NOT NULL default '0000-00-00 00:00:00',
```

```
-> duration int(11) NOT NULL default '0',
-> billsec int(11) NOT NULL default '0',
-> disposition varchar(45) NOT NULL default '',
-> amaflags int(11) NOT NULL default '0'
-> );
```

That's all there is to it! We only have to do this once, so it's really not so bad. Next, we have to modify the `/etc/asterisk/cdr_mysql.conf` file to correctly reflect our choices.

```
[global]
hostname=ourdbserver.ourdomain.tld
dbname=asteriskcdrdb
password=changethis2yourpassword
user=asteriskcdruser
port=3306
userfield=1
```

The next time we restart Asterisk, our CDR information will be stored in the MySQL database. What does that give us? We now have the ability to use a number of very powerful tools to search our CDR records to find trends and patterns.

Monitoring Calls

Slightly less friendly than recording the information about a call is enabling the ability to monitor calls in real time. This allows us to listen in to a conversation as it happens, so that we may see how our customers are being treated.

This feature is only available on Zaptel channels. The application to use to monitor the channel is called ZapBarge. It can only accept one command line argument, which is the number of the channel to listen in on. If we do not pass ZapBarge an argument, it will prompt us to enter one. The channel numbers it requests are the same channel numbers given in `zaptel.conf` and `zapata.conf`.

Suppose we had four outgoing Zaptel channels, numbered 1 through 4. We could have something like this in our `extensions.conf`:

```
exten => 8700,1,ZapBarge
exten => 8700,2,Hangup
exten => 8701,1,ZapBarge(1)
exten => 8701,2,Hangup
exten => 8702,1,ZapBarge(2)
exten => 8702,2,Hangup
exten => 8703,1,ZapBarge(3)
exten => 8703,2,Hangup
exten => 8704,1,ZapBarge(4)
exten => 8704,2,Hangup
```

This way, extension 8700 would give us access to any Zaptel channel; whether it is an FXO or FXS interface does not matter. Then, extensions 8701 through 8704 would give access to each of the outgoing interfaces.

While these extensions are useful, there is a danger, and we must consider security. Clearly, we do not want just any employee to be able to listen to calls that are in progress. And more than that, we really don't want our customers to be able to accidentally listen to calls that are in progress.

A good way to handle this problem is to create a separate context just for monitoring extensions. Then, designate a single telephone handset that will be able to do nothing but monitor extensions. This handset should be the only phone in the monitoring context, and the monitoring context should not be included in any other context. Keep that handset under lock and key. Not only will this keep from people overhearing embarrassing or confidential information, it will also go a long way towards fostering trust with the employees.

Recording Calls

The last of all of the Quality Assurance methods we will discuss is the call recording capability of Asterisk. This is highest on the Big Brother chart because phone conversations can be archived forever and reviewed on demand. Therefore, an employee's entire telephone history can be called up at any time.

This feature can be accessed from a number of different sources. First, we can configure specific call queues to record calls. This is done in the `queues.conf` file, for each individual queue. We set it thus:

```
[100]
. . .
monitor-format = wav
monitor-join = yes
```

The first line tells Asterisk to record the conversation in the `.wav` format. This is the best choice because it is most compatible with other operating systems. Since archived conversations can be burned to CDs, compatibility is a high priority. The second line tells Asterisk to join the two files (in and out) into one file. If we do not do this, we will only hear half of the conversation. To take advantage of this feature, we must have `soxmix` installed on our Asterisk server. The Red Hat packages that contain `sox` are missing `soxmix`; therefore, to install `soxmix` on Red Hat Linux, we need to do so from source.

All calls coming into the queue will be recorded. The name of the file will be the unique ID that Asterisk generates for every call. If we wish to change this, we can do so by adding something like the following in `extensions.conf`:

```
exten => 100,1,SetVar(MONITOR_FILENAME=${DATETIME}-${CALLERIDNUM})
exten => 100,2,Queue(100)
```

This will record all calls coming through the queue named 100, in `.wav` format. We then are free to encode them into MP3 format if we wish to save space.

Aside from recording calls in queues, we can also monitor arbitrary calls through the use of the dialplan. The name of the application that records a channel is `Record`. To start recording, we call the application like this:

```
exten => 200,1,Record(${TIMESTAMP}${CALLERIDNUM}-${EXTEN}.wav)
exten => 200,2,Dial(SIP/1001)
```

With just one line of code in our dialplan, we can start monitoring calls. If we want, we could even insert this line into our macro definitions for standard extension types. Or, we could do something like:

```
[incoming]
exten => _.,1,Record(${TIMESTAMP}${CALLERIDNUM}-${EXTEN}.wav)
exten => _.,2,Playback(thiscallmaybemonitoredorrecorded)
exten => _.,3,Goto(default,${EXTEN},1)
```

In three lines of code, we have enabled recording for all incoming phone calls. We have even notified our customers that the call may be recorded. We have the power. Should we use it?

Legal Concerns

> This is not legal advice. Only a qualified attorney can advise you on your particular situation.

It is very important to note that, just because we can monitor calls, doesn't mean we should, or even that it would be legal to do so. Many states in the United States of America are 2-party or all-party states, meaning that all parties to a conversation must know that a call is being recorded for it to be admissible in court.

More than that, there are privacy laws in place to protect everyone. Only a careful study of all applicable laws can tell us if we are in the clear. We should never record any phone calls until we have spoken to a lawyer.

But aside from the legal issues, there are also moral issues. Maybe it depends on our intent when we call. Are we recording the calls to help our employees improve? Are we recording the calls so that we have an accurate representation of what was agreed upon? Or are we recording calls to try to trap someone, or to pull information out of calls to be used out of context later?

Summary

As we have seen in this chapter, Asterisk gives us the power to:

- record call information
- monitor conversations
- record the conversions themselves

The purpose of these capabilities is to provide us with options for using our system effectively. It is our responsibility to use these powers appropriately.

There is no point recording all calls if you are never going to use those recordings. Similarly a database is overkill if you have no real interest in your calling history.

However there are many reasons to use these features, for instance to produce reports or answer questions that other users or departments have regarding the telephone system. The users of the system will know more about what they will need in order to carry out their day to day duties, which is why we spend time figuring out exactly what they need early in the deployment plan: to ensure the system provides everything that is needed.

[PACKT PUBLISHING]

7
Asterisk@Home

As we discussed when introducing Asterisk, flexibility is a primary focus. Asterisk can be used for a variety of different purposes and each available feature can be tailored to the specific needs of any organization. Asterisk@Home retains some of the flexibility and adds a massive amount of convenience and ease of use. It offers web-based configuration, web access to voice mail, reporting, and other functions, which we will cover shortly. All these functions can be added to your own Asterisk installation as they are all based on existing Asterisk tools; the real benefit of Asterisk@Home is that you don't have to set these up manually to make them work together. Asterisk@Home installs and sets up the base configuration for all of the tools it provides.

It's extremely easy to set up and use; the downside to this is that you lose some of the flexibility—you can't choose which OS to base your Asterisk system on, for example, and you can't fully customize the configuration, which becomes a hindrance in larger Asterisk installations.. Its name, however, understates its scope; there have even been discussions to change it because of its confusing nature. Its functionality goes quite a bit beyond that needed by a home user and it can be easily set up to handle a small-to-medium-sized business's calling needs. However, beyond that size, it becomes a bit harder to scale effectively than a plain Asterisk install.

CentOS

Asterisk@Home is designed around the CentOS distribution of Linux. CentOS is built from the Red Hat Enterprise source packages. It has a relatively small core team of developers that concentrate on packaging the OS without Red Hat's proprietary components. The main focus of CentOS is to provide a freely available operating system with the packages and features needed at enterprise level, without the cost associated of the base distribution, Red Hat Enterprise Linux. CentOS does, however, offer a range of commercial support, which is invaluable to most enterprises and thus is an option we can consider.

CentOS isn't the focus of this chapter and it doesn't really have too much bearing on our use of Asterisk@Home other than knowing basically how to use and update it. We will focus on the setup and maintenance of Asterisk@Home and the features it provides for us. It would be beneficial if we spent time getting to know CentOS if we decide to use Asterisk@Home.

Preparation and Installation

Asterisk@Home recommends a minimum of a 300Mhz processor and 128 MB of RAM; however, CentOS will complain at boot if there is less than 256 MB. The amount of RAM required has a direct correspondence with how heavily used the system will be. As Asterisk@Home is Asterisk with a few other services added, we can pretty much scale it similarly. We do have to consider extra resources for the additional services we have running such as the web server and the MySQL server.

Asterisk@Home comes in two flavors: a source package that we can install on a CentOS system, and an ISO image that can be burned and installed as a full OS. The ISO installs a modified CentOS system automatically and sets up the necessary Asterisk@Home services.

We can obtain the ISO from `http://sourceforge.net/projects/asteriskathome/`.

After downloading and burning the image to disk, reboot the target machine with the Asterisk@Home CD in the drive, and wait for the prompt, which should look like the following:

```
boot: _
```

If at this point we hit *Enter*, the installer will start up and begin to install CentOS with Asterisk@Home on the first primary hard disk. It's very important to ensure that this disk is the disk we want to use and that no important data is held there as all data will be lost. From this point onwards installation is entirely automatic, and we can leave it for a few minutes while it prepares the machine, installs the OS and the necessary programs for Asterisk@Home—including Asterisk, MySQL, Apache, and so on. It's a good time to gather the documents we need to configure Asterisk, such as our lists of extensions and our service provider account details. We'll need the same information as in previous chapters where we set up Asterisk manually; now, however, we don't need to worry so much about Asterisk's configuration syntax as we have a friendly GUI-based setup system that takes care of most things.

Installation of Asterisk@Home is extremely simple and as long as all of our hardware has Linux support, there should be little issue getting the system installed.

We can configure advanced options and modify the kernel boot parameters if necessary by hitting one of the keys *F1-F5* at the boot prompt (this usually isn't necessary). *F5* is of particular note as this runs the CD as a rescue disk, which we can use to repair a machine that refuses to boot.

Also, if we have problems getting the OS installed from the CD we can enter `linux mediacheck` at the boot prompt to confirm the integrity of the installation disk, something worth doing to ensure that the ISO was burnt properly rather than waiting for it to break during installation.

When the system has finished installing, it should reboot and leave us at the login prompt. We should log in as root and change the default system passwords. The default root password is "password". The commands we need to run are:

```
# passwd          ;to change our root password
# passwd admin    ;to change the admin password for SugarCRM
# passwd-maint    ;to change the maint user's password for the Asterisk
                   Management Portal
# passwd-amp      ;to change the amp user's password
# passwd-meetme   ;to change the Web MeetMe user's password
```

It's important that we change these passwords as soon as possible to ensure we don't deploy the system with default passwords with the risk of a possible security breach.

After we have the passwords set, it's time to ensure we can access the machine over the network: firstly we should check if the machine has picked up an IP address at boot. We should see at login if we have an IP, but to be absolutely sure, we should run `ifconfig` and check that an IP address has been assigned to our required network interface.

```
# ifconfig eth0 | grep "inet addr"
```

This should show a line containing our IP address. If no IP address is shown or we want to set a static IP for the Asterisk box, which is often more useful, then we can run `netconfig`. In this case, we will have to input the network settings for the machine, namely the IP address we want to use, the subnet mask, default gateway, and DNS server.

```
# netconfig                    ;Enter IP details
# /etc/init.d/network restart ;Apply the changed settings
```

IP Addressing

As we may have SIP clients and as we access Asterisk@Home using a web browser, it is usually beneficial to have the Asterisk@Home machine configured with a static IP, or if we are using DHCP to ensure the address is reserved so that it doesn't change. This ensures we can always find the server without reconfiguring clients.

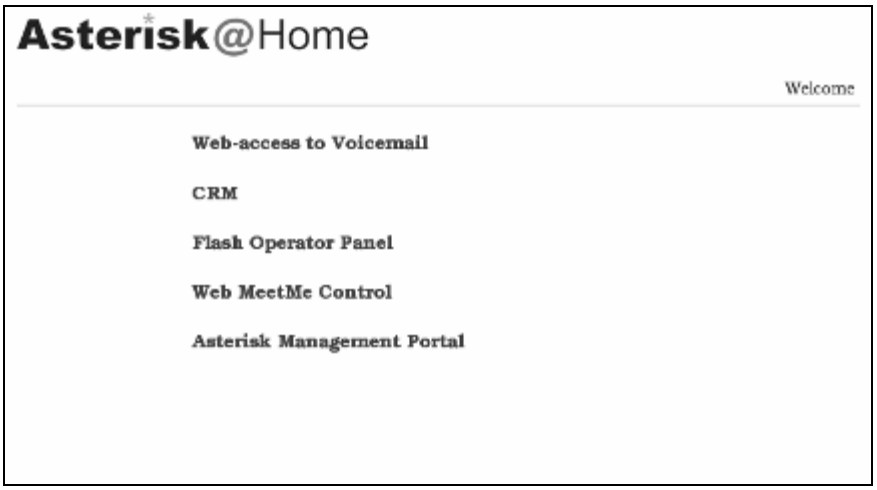
The Asterisk Management Portal (AMP)

Now that our system is installed, base passwords are set and we have network connectivity, we can begin to configure the server to perform its role. This is done by using the web management utilities the system provides and in some cases, when necessary, modifying the underlying Asterisk configuration files.

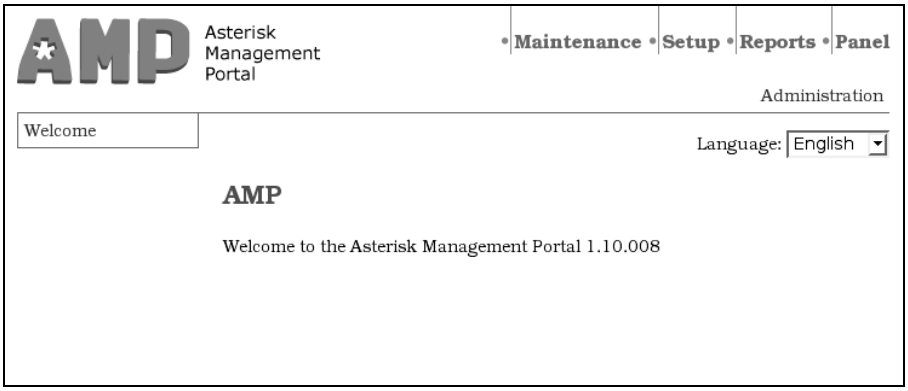
We will take a look at the functionality provided with the web interface and then follow an example setup, which will create a working Asterisk@Home server with a single PSTN line and a single SIP extension.

The first place we want to look is most likely the management portal, which allows us to configure most features of the Asterisk system in an arguably eye-pleasing and user-friendly fashion. With this interface, we can manage everything including extensions, ring groups and trunks, our MySQL database, and even report generation by the system.

To get to the management portal, open a web browser and type the following URL in the address bar: `http://<AsteriskIP>/` (use the IP address we set earlier) you should be presented with a screen like this:



If you click on Asterisk Management Portal (or go there directly with `http://<AsteriskIP>/admin/`). We will be prompted for a password to which we will reply with user maint and the password we previously set for this user. We will now be presented with four menu options at the top right-hand corner, as shown below:



Maintenance

Unsurprisingly this section is where we can maintain the system, check the status of services and access the back-end database, for example.

- System Status

  This page shows the current status of the system. It should show whether or not Asterisk, cron, secure shell, and the web server are running and give us the option of rebooting or shutting down the machine.

- Cisco Phone

  This page gives us access to the configuration for Cisco IP phones. We can add, edit, or delete the automatic configuration scripts for these devices.

- Config Edit

  This section lets us access the configuration files for Asterisk and CentOS itself via a web-based editor. For example, we can edit `Asterisk.conf`, `resolv.conf`, and any other files in the `/etc` directory. This becomes useful when we find an area of the GUI that doesn't fit our requirements for customizing the system and have to edit the relevant configuration file by hand.

- PHPMyAdmin

  This gives us access to the web-based MySQL management tool PHPMyAdmin. This tool is extremely useful for Asterisk@Home since much of the configuration and logs are held in a MySQL database. We can back up the databases, run SQL queries to view or modify the existing databases, and even add databases of our own.

- Sysinfo

  This page gives an overview of the current system state—covering network, memory and hard disk utilization as well as some system specifics.

- Asterisk Info

  This is also overview information, but focusing on Asterisk itself. We can see information about SIP, Zaptel, and IAX usage as well as version and system information for the Asterisk service.

- Webmail

  This gives access to the built-in webmail system, which is also useful if we use the server to store our voicemail as email attachments. It can be used even if we don't email the voicemail attachments.

We can also use the maintenance section of the interface to view system logs, to upload audio files for use by Asterisk, and to download backups of the system.

Setup

Under this section, we will find the relevant pages for configuring our extensions, lines, trunks, conferences, and other Asterisk features.

- Incoming Calls

  Here, we can configure how incoming calls from the PSTN are handled. We can route them to a receptionist, extension, ring group, or queue, and optionally have this change automatically at specified times, such as outside working hours.

- Extensions

  To add an extension we would use this section. The information we would provide would be the protocol in use (e.g. ZAP), the extension number, password, the user's full name, and whether or not we would like to record incoming and/or outgoing calls for this extension.

  This obviously is simpler and more intuitive than modifying the Asterisk configuration files directly, although we do lose a little bit of flexibility in how these extensions are added. We can make up for this with some hand-hacking as required. We can also configure voicemail for the extension here if we wish.

- Ring Groups

  If we need a group of extensions to act together, as covered in a previous chapter, we need to set up ring groups. This section lets us create groups and detail the extensions that we would like to be in these groups.

- Queues

  If we expect largish volumes of calls, then we will need to queue them and this section is where we configure our queues. To configure a queue, we provide the ID, name, password, CID prefix, and available agents. We can also add on-hold music and set other queue options such as announcements and their frequency.

- Digital Receptionist

  A digital receptionist can be used for fail-over when a queue is at capacity or an extension isn't available. We can record greetings for this here and set it up on the required extension. These receptionists can also obviously be dialed directly, if required as information systems for example.

- Trunks

  Allows us to add a variety of trunk types such as SIP and IAX to our system much as we did in previous chapters. A nice wizard-based system will prompt us for all the necessary parameters such as trunk type, name, DIDs and so on.

- Outbound Routing

  We should now be aware whether calls to different places should be routed over different providers. For example we may have a local service and a VoIP server where we use VoIP for international calls and the local service for local and national calls. In this section we can configure dial patterns to ensure we route calls over the most efficient and cost-effective line.

- DID Routes

  We can easily associate our specific DIDs to their corresponding extension or other service.

- On hold Music/System Recordings

  We can record, upload, and manage our various audio files for use with the system.

- Backup and Restore

  We can make one-off backups as well as complete restores of the system. This may be slightly useful but a fully implemented backup strategy, managed outside of the AMP interface, would be more beneficial to us.

- General Settings

  Here we can configure settings such as the number to dial for an outside line and a fax machine extension or email address.

- Reports

  As well as setting up and maintaining our system we may also require various reports. Here we can view reports by date and we can view full call logs, and compare calls, as well as monitor monthly and daily traffic. This function is provided by the Areski Asterisk-stat tool (`http://areski.net/asterisk-stat-v2/`).

Flash Operator Panel (FOP)

This panel is extremely useful as it shows us all extensions, conferences, and queues with details of their status. We can use this to get a current overview of system usage. It's a Flash-based real-time interface to the system state. It can also be used to hang up, transfer and originate calls via drag and drop as well as provide "pop-up" functionality where the customer's details appear on screen according to their CLI details. All this can be protected so as to restrict agents' access to every function of the panel.

It's quite an intuitive interface, so most actions are taken with button clicks and mouse movements; for example dragging a free channel to a bridged channel will allow us to barge into the existing call.

Flash Operator Configuration Files

The FOP can be configured by editing the configuration files that are shipped with it. They can be found in the /var/www/html folder and can also be accessed from the Asterisk Management Portal.

The files include:

- `op_astdb.cfg`
- `op_buttons.cfg`
- `op_buttons_additional.cfg`
- `op_buttons_custom.cfg`
- `op_server.cfg`
- `op_style.cfg`

`op_server.cfg` is the most important for initial setup. Here the main FOP configuration lives, including the IP address of the Asterisk service, the username and password for accessing FOP, as well as any debugging settings that should be applied. You can also configure your available conferences here for example. The other files can be used to add customized settings such as extra buttons for the system and modifications to the style of the FOP.

Web MeetMe

Web MeetMe is a web-based front end to the MeetMe add-on for asterisk. It allows us to monitor and control conferences. By clicking on Web MeetMe Control on the Asterisk@Home web interface, we are taken to the interface for this program. We access it by inputting the conference number, taking us to a screen that lists all the participants in the conference. We are made aware of the caller's name, user ID, channel, and which mode they are in, such as "listen and talk" or "listen only". We can also modify these modes by clicking a link, and we can remove a participant from the conversation entirely. This sort of control is extremely handy for users not comfortable controlling a conference from their handset, as the visual cues and graphical interfaces to commands make it extremely easy to get to grips with.

Flexibility When Needed

We have looked at a few graphical configuration tools that add a lot of convenience and ease of use to the Asterisk system. As with any GUI, the focus is clarity, ease of use, and intuitive design. When we take a powerful command-line or service-based application and add a GUI to it, there is often a loss of flexibility. As Asterisk holds flexibility as one of its most important aspects, this may seem like a major downside to Asterisk@Home.

However, we can still get under the skin and make up for some of the shortcomings in the graphical interfaces. As we have seen, the Asterisk Management Portal provides a direct link to the text configuration files; a testament to the fact that the GUI is merely a layer upon a powerful underlying system.

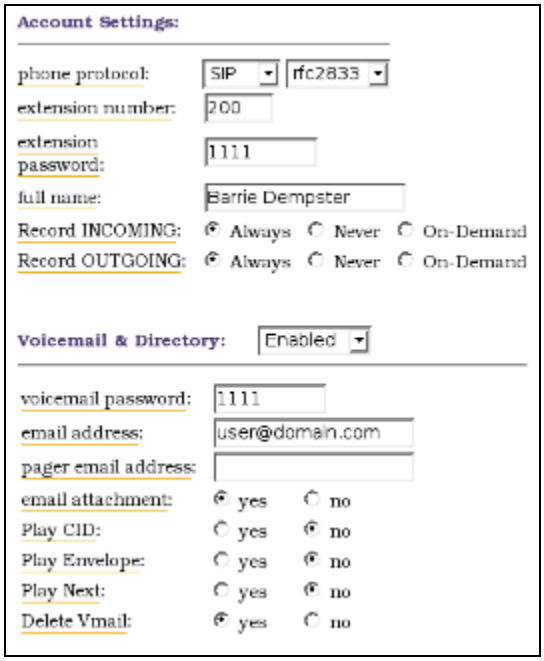
If we find that there are inadequacies in the GUI for us, then we can edit these files by hand in order to get the functionality we need. There is one major caveat with this, however: we must ensure that we are attentive to the automatic settings produced by the GUI and ensure that any alterations we make are going to be compatible with the GUI, or else we risk breaking the interface entirely; this can be quite a hindrance if our system is to become complicated. This still restricts us—we can't tailor Asterisk@Home to the same degree as we can with Asterisk—and so it isn't always the best option. If we have trouble customizing Asterisk@Home to our need, then creating our own Asterisk build may be a more viable option.

A Simple One-to-One PBX

Now that we have an overview of how the main features of the Asterisk@Home system are customized, we can create a simple PBX for handling a single line and extension for a home user. We can also take the knowledge of call routing gained from previous chapters and apply it to Asterisk@Home: all of the concepts remain the same, we just apply them differently, and the result is virtually indistinguishable.

Extensions

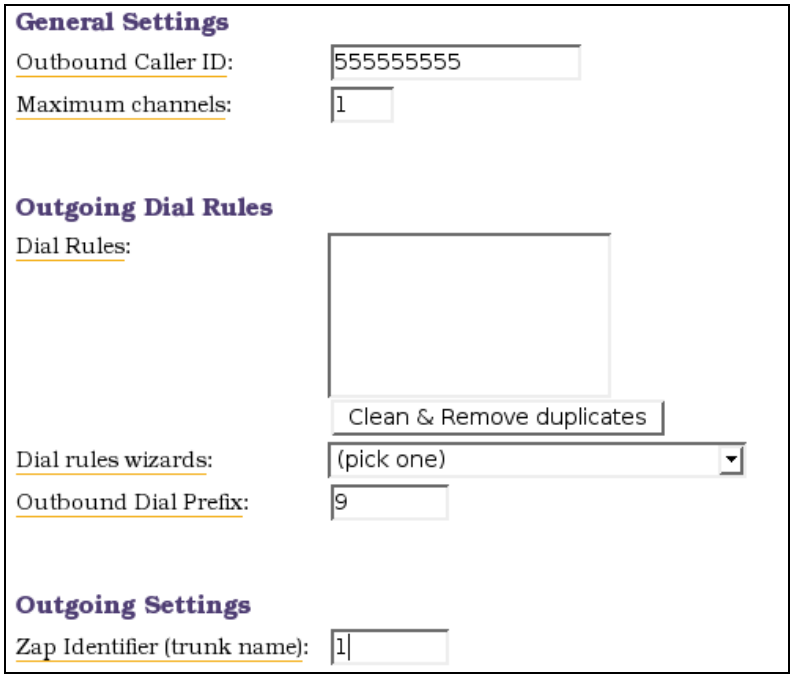
Firstly we will configure our extensions, by opening up the AMP. Click on Setup and find Extensions on the left-hand side. Then configure the extension screen as follows (you may wish to change some settings to fit your own needs):



After we have configured the extension, click Add Extension on the bottom right of the screen. This sets up extension 200 for a SIP based phone. We need to click the red bar that appears afterwards to apply the changes to the system.

Trunks

We can now add the trunk for our PSTN interface. We do so by clicking Trunks on the left-hand side, clicking on Add ZAP Trunk (we can add other trunk types as discussed in previous chapters, such as SIP and IAX), and then configuring the trunk as follows (we may also want to delete the default trunk `g0` while here):

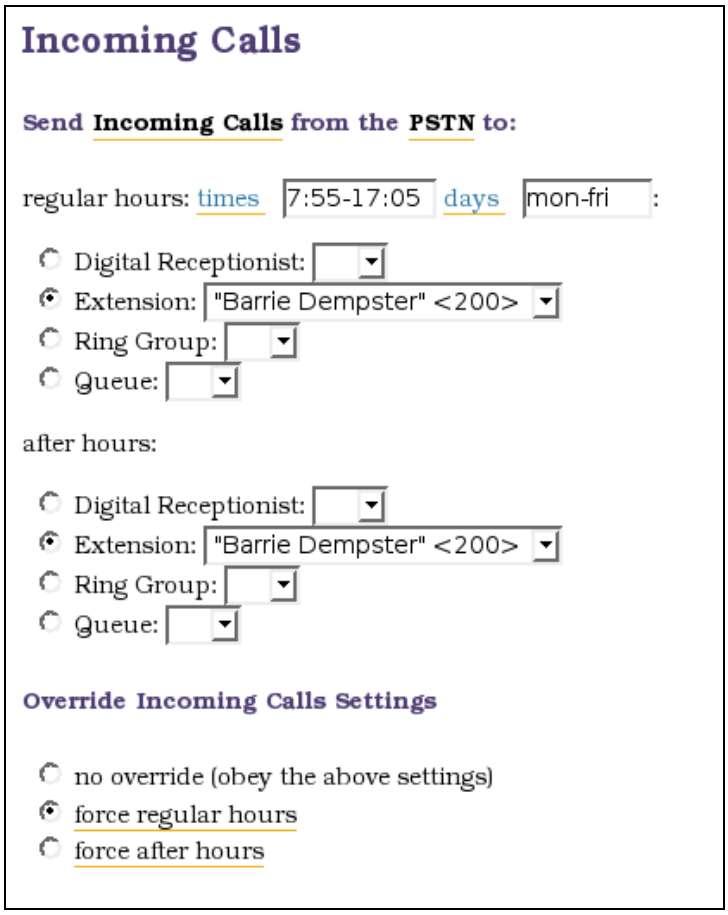


Obviously we would replace the 555555555 with our own phone number. Again remember to click the red bar afterwards. It's important to note that when we make calls, there is often no check made against the caller ID number we present so we could present anything here. We must verify that it's completely accurate or we may lose the ability for our contacts to recognize us and call us back. This can often be used to an advantage when we want to control the number we present.

Routes

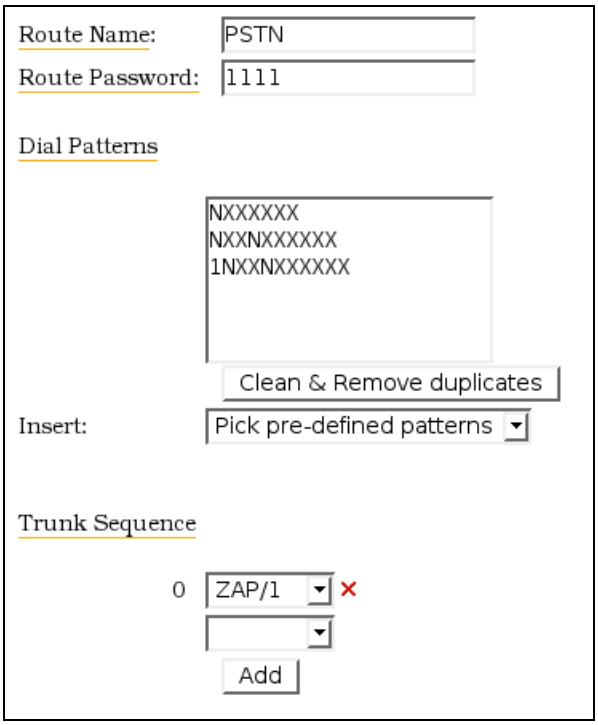
Now that we have extensions and trunks, we require incoming and outgoing calling routes so that calls get to their correct destinations.

Firstly create an incoming route by clicking on Incoming Calls and configuring it as follows:



Then click Submit Changes and our little friend the red bar to confirm.

We also require outgoing routes so that we can route our calls through the trunk that we have set up. Do this by clicking Outbound Routing and then configuring it as shown n the screenshot that follows. You can modify the dial pattern here and can add alternative routes with differing patterns.



We should now be able to make and receive calls from our system over the PSTN. We should also have working voicemail.

Customer Relationship Management/SugarCRM

Also provided with Asterisk@Home is a Customer (or Contact) Relationship Management system (CRM), which can manage a communication relationship with a contact. As we live in a society with multiple levels of communication and we have many conversations per day by fax, email, telephone, face to face, and so on, managing and collaborating these can be a nightmare. This is where a CRM system becomes useful. We can take all these calls, emails, text messages, faxes, and other methods and manage them within a single system. SugarCRM allows us to do this and while doing so integrates with Asterisk@Home and its calling features.

With SugarCRM, we can manage our contacts, calls, and tasks and make others aware of what we are doing to share our current state of communication on a particular topic, task, or customer. "Has Bill in sales called that customer yet?" "Did Jane send Tom the status reports?" Questions like these can be answered easily with a CRM system. In this section, we take a look at a few of the features that SugarCRM provides.

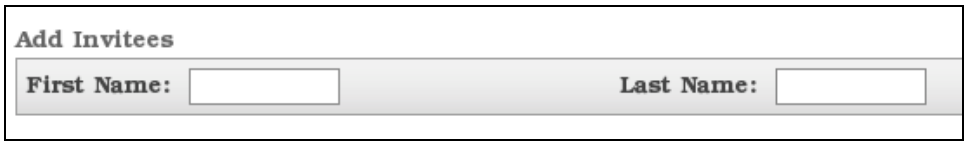
Adding Contacts

By clicking on the CRM link on the front page of the Asterisk@Home web server, we are taken to a login page where we can log in as the user admin with the password we set for this user previously. On the first screen, we will see any upcoming tasks that we need to attend to along with anything else that's currently outstanding. From here, we can move around and add contacts, tasks, and so on to the system.

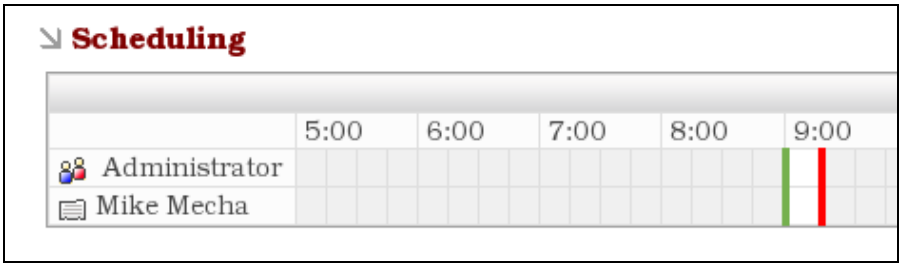
If we click on Create Contact, we will be taken to a very large screen with a massive number of empty boxes for us to fill in; luckily not every field is required and we only need to add a surname at minimum, although the system does not become useful until we add some contact details. We should create one contact with at least an email address and telephone number for now, so that we can try out the other functions on offer.

Call Scheduling

Now that we have a contact in the system, we can schedule calls and appointments, and send emails to them. After adding the contact, we should be at a contact screen, where we can view the details we just entered. Click on the schedule call button (this doesn't schedule an automatic call; it schedules a reminder for the user to initiate a call). You should be presented with a screen asking for details of the planned call. Fill in the subject field with some text such as "test call", and then enter the details of the contact we added previously at the bottom of the screen where we see:



Click Search and find the contact's name below the Add Invitees box and click add, which is on the right-hand side. You should then see the name added to the attendees list along with "Administrator".



When done, click Save, which adds the call to the database and it should appear on your home screen within SugarCRM.

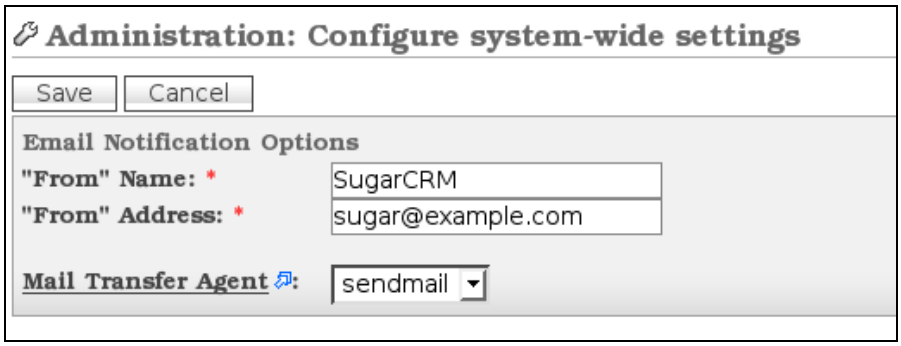
Setting up tasks and meetings is a similar process and we also have the option of emailing invites to the events to the specified attendees. Similar functionality to that provided by Microsoft Outlook/Exchange, but with the added advantage of being much more focused on customer relationship management rather than merely scheduling your own time. It is extremely useful for groups of people working together on the same or similar accounts.

Administration of SugarCRM

Now that we have a basic overview of some of the functions SugarCRM provides, we can have a look at how we administer the back end of the system. By clicking admin, we are taken to the section of the system where we can make changes to the default settings, manage user accounts, and even customize the forms on the other parts of the system. We will briefly look at the basic administration settings and user management.

Configure Settings

Under this section we see a small area that contains email settings:



In this section, we could change the name and address emails come from to reflect that they are sent by our system:

"From" Name: Company CRM Administrator

"From" Address: administrator@ourcompany.com.

User Management

Here we can add, edit, and remove users of the SugarCRM system. Clicking Create User shows a page where we can fill in various details of our system's users. Again this is a wide-reaching page with a variety of options, not all of them necessary. If we click on the existing Administrator account, we see a similar page showing all the attributes currently set on the Administrator account.

When viewing a user account, we can click on the edit button showing a similar screen but this time the fields are editable and we can modify any attribute we feel necessary. We can also delete the account, although it is recommended to disable accounts of users that have left the system. This is so that any previous history assigned to them is still linked to their account, and also in the event that the user comes back for any reason the account can be recreated as before with minimal effort.

We can now add users that will be using the CRM system and modify their attributes appropriately.

User Roles

As our system is being used, it may be necessary to segment usage of the CRM so that users with varying responsibilities have different roles in the CRM and appropriate permissions to complete their job functions. By clicking on Create Role we are presented with a page where we can define a new role for use with the system. We need only provide a name and then provide the modules users in this role are allowed to access.

For example a junior salesperson may not be allowed to create leads and appointments but a more senior employee may. In this scenario, we would create two roles and define access to modules accordingly.

Summary

Asterisk as a system is extremely flexible as we are now well aware. However the addition of a powerful tool such as SugarCRM makes it more than simply the telecommunications provider for the organization. It becomes the focal center of all communication, be it incoming, outgoing, or internal.

We can use Asterisk and this CRM combined to plan entire projects and monitor their progress from start to finish. We can plan, make, and receive emails, calls, faxes, and meetings. We can determine how long we spent on tasks towards a specific account or project and we can see who was involved in all the conversations.

This paves the way for a variety of reports and analysis on how our business runs, which just isn't easily possible without the integration and flexibility of our open-source telecommunications system. This added functionality is exactly what the Asterisk system is designed to do. It excels as the powerful back-end technology running behind an integrated and feature-rich system.

The flip-side to this sort of integration is that all our eggs are in one basket—we have no distribution of data. This can be a major problem to us as our Asterisk system expands. The functionality of Asterisk@Home is available in Asterisk with the required add-ons (AMP, FOP, etc.). With our own Asterisk build we can host some of these services on different machines, therefore spreading the load and the data. As mentioned earlier, Asterisk@Home is great for smaller installations as an excellent "plug and play" telephone system. If we expect significant growth or are starting with an already medium to large organization then we should consider building our Asterisk system from scratch.

[PACKT PUBLISHING]

8
Case Studies

Up to this point, we have worked through the setting up of a new phone system based on Asterisk. Our system is developed with our specific needs in mind and has configurations and features enabled to fit our particular purposes. Often we will find ourselves wondering if we're making full use of the magic of Asterisk, and perhaps the best way to find new tricks is to examine examples of working phone systems.

What follow in this chapter are typical examples of a few just such systems. Each section will be devoted to one type of system. First, we will give a brief overview of the type of customer involved before we mention some pointers to remember as we decide what they need and how best to accomplish our task. Finally, we will go through the configuration files, one at a time. Each configuration file will be annotated, and we will briefly discuss why some of the choices are particularly good for the given scenario.

Small Office/Home Office

This is a common setup for Asterisk. In very small installations, Asterisk can be used to give us the features we need from an expensive PBX system at small office prices. Using Asterisk as a phone system for a home office gives the small business a big-business sound and feel.

The Scenario

For our first example, we will join David in his home. An avid programmer and all-round geek, he's decided he wants to have his very own phone system in his house. Because he is too cheap to buy an answering machine, the system must have voicemail. He has recently moved into an older home and only has one incoming line from the telephone company.

David has a new baby in the home and is very concerned about safety. He lives out in the middle of nowhere, and loses power pretty regularly. He only has phones in the office, kitchen, and master bedroom.

Finally, David would like to have Music on Hold for people who call in. Since he is starting a new business, he wants to present a professional image to the callers he places on hold for hours on end while he finishes his Pot Noodle.

The Discussion

First, Asterisk could be an appropriate choice here. Since David is a programmer, he will be comfortable managing his own phone system. For home users who are not technically minded, Asterisk may not be a good fit, unless they will be supported by a larger IT department at their employer.

Secondly, to meet David's requirements, we will have to use POTS lines. Since he moved into an older house, running cable will be very difficult. Also, since he lives in an area prone to power failures, having the Asterisk server provide power to the telephones is a good thing, as this will require only one UPS.

Giving David voicemail and Music on Hold will be very simple. In fact, the thing to notice here, as with many SOHO configurations, is the lack of any other requests. This means we will not need to configure SIP, IAX, H.323, parking, menus, or any other advanced feature.

Since he has three handsets and one incoming line, we will be using a Digium TDM31B card. This is a hardware TDM device that has three FXS ports and one FXO port. Also, since he has such low requirements, i.e. all calls will be directly bridged on TDM interfaces, he will be able to add Asterisk to his email server, which is already running Linux on a Dell PowerEdge 400 SC.

The Configuration

Here are our configuration files for David. We should note that his server has been set up as we have previously discussed, including mpg123. Also, his house has home-run telephone wiring, meaning all telephone jacks terminate to the same closet.

Any files not listed here are left as default. If we did not install the default files, we may do so at any time by changing to the `/usr/src/asterisk` directory and executing `make samples`.

zaptel.conf

```
fxoks=1-3  ; 3 fxs modules, on channels 1-3
fxsks=4    ; 1 fxo module on channel 4
loadzone=us
defaultzone=us
```

We should remember that FXS modules use FXO signaling, and FXO modules use FXS signaling. Also, we should always put our FXS modules on the lower port numbers because of some reported inconsistencies when putting an FXO module on channel 1 of a TDM card.

zapata.conf

```
[trunkgroups]
[channels]
language=en
usecallerid=yes    ; Even though David is cheap, he subscribes to CID
hidecallerid=no
callwaiting=no     ; but not call waiting...
usecallingpres=yes ; NOTE: this does not always work right, but when
                   ;  it does, it is quite useful
transfer=yes
cancallforward=yes
callreturn=yes
echocancel=yes
echocancelwhenbridged=no
echotraining=800   ; he had echo until he set the train time to 800

; FXO Interface
context=default    ; all calls go to the "default" context
signalling=fxs_ks  ; we use FXS signaling for our FXO device
group=1            ; we are placing the outgoing line in a group
channel=>4

; FXS Interfaces
context=outgoing   ; All phones in the house may dial long distance
signalling=fxo_ks  ; we use FXO signaling for FXS devices
group=2            ; we are putting all internal phones in a group
pickupgroup=2      ; this is so we can pick calls from other lines
callgroup=2        ;  which may not be useful in this instance, but
                   ;  should not hurt anything
channel=>1-3       ; We select the channels
```

A few things we should notice: first, we usually don't want to enable echo cancellation when calls are bridged. This can especially cause a problem with modem and fax communications. Also, we can modify the echo training period to the best value for our particular installation using a process of trial and error.

Another note is that we are not segregating the phones based on where they can call. It may be tempting to put all of the incoming and outgoing extensions in a single context; however, it is not wise to do so from a security standpoint. Thus, we have a blend of security and simplicity.

musiconhold.conf

```
[classes]
default => quietmp3:/var/spool/asterisk/defaultMOH,-z
```

Here we have our Music on Hold configuration. Notice that we only have one class, which we called "default." Also, we have chosen to shuffle our files so that the system doesn't always start with the same song. In small installations, it is very important to remember this, as it is not very impressive if every time a customer calls, they are greeted with exactly the same Music on Hold selections.

voicemail.conf

```
[general]
format=wav49|gsm|wav
serveremail=asterisk@davidscomputer.com
attach=yes     ; voicemail messages will be attached to emails
skipms=3000
maxsilence=10
silencethreshold=128
maxlogins=6    ; David's wife isn't so good with passwords...

[zonemessages]
central=America/Chicago|'vm-received' Q 'digits/at' IMp

[default]
1 => 1234,David
Gomillion,david@mydomain.tld,pager@mydomain.tld,tz=central
```

We have only one time zone, which is Central in the United States. We also have only one voicemail box. Asterisk can do a lot more, but in this instance, no more is needed.

The configuration choices we made at the beginning of the file are pretty much standard, except for the server's return email address. This should be set to something meaningful if we are going to have users who will reply to these messages; however, in this instance, this is just a fake (but informative) address because David simply won't try to reply to these email notifications.

modules.conf

```
[modules]
autoload=yes
noload => pbx_gtkconsole.so  ;don't load stuff we won't need
noload => pbx_kdeconsole.so
noload => chan_sip.so
noload => chan_iax.so
noload => chan_iax2.so
noload => chan_skinny.so
noload => chan_mgcp.so
noload => res_agi.so
noload => app_intercom.so
load => chan_modem.so
load => res_musiconhold.so

[global]
chan_modem.so=yes
```

In our `modules.conf` file, we have disabled all of the VoIP protocols that we will not be using. This will help increase the security of our server, as this keeps ports closed that have no need to be open. We also have firewalled all ports on the server except those needed for other servers.

extensions.conf

```
[general]
static=yes
writeprotect=no

[globals]
TRUNK=Zap/g1
TRUNKMSD=1

[outgoing]
exten => _9.,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}}) ;if we dial 9,
                                                    ; send to trunk
include => default

[default]
exten => s,1,Dial(Zap/g2,30)         ; dial all extensions for 30
                                     ; seconds
exten => s,2,Voicemail(u1)           ; send to VM if we don't pick up
exten => s,3,Hangup
exten => s,102,Voicemail(b1)         ; send to VM if we are busy
exten => s,103,Hangup

exten => 0,1,Dial(Zap/g2)            ; if we dial 0, ring all phones
exten => 1,1,Dial(Zap/1)             ; if we dial 1, ring the office
exten => 2,1,Dial(Zap/2)             ; if we dial 2, ring the bedroom
exten => 3,1,Dial(Zap/3)             ; if we dial 3, ring the kitchen
exten => 8,1,VoicemailMain(s1)       ; press 8 to check messages
                                     ;  without requiring password

exten => i,1,Goto(s,1)         ; if we are in an invalid or timed-out
exten => t,1,Goto(s,1)         ; state, go to s,1 in this context
```

This is our entire dialplan. We can see that it is very simple: each phone has an extension, and there is an extension for all phones. Only incoming calls are going to go to voicemail if a phone is busy or not answered.

We will notice that any number that is dialed with 9 as the first digit will automatically be sent out the trunk. This is a very simple example of how a single pattern can accomplish many tasks. Since we are not very concerned about securing the trunk from internal extensions, it is alright to use this simple method of trunk access.

Conclusions

As we can see, Asterisk configurations can be very simple. Creating a PBX for a few extensions is easy. Moreover, it illustrates some points that we will also see later in configuring some other PBX systems.

Small Business

Small businesses make up a large portion of the IT market. These customers are unlike any other: they need upscale features with limited resources. It is very common for small businesses to require advanced features while needing to keep costs down. It's also common for small businesses to want to appear larger and more established than they are to increase customer confidence. Asterisk can be a great solution for small businesses as it suits these needs well.

The Scenario

ACMEsoft is a software engineering firm with 40 employees. According to recent usage studies by their telephone company, they usually use about 18 lines, with their peak last month at 22 lines. They have a number of hosted extensions from the local telephone company (often referred to as Centrex service), which they have been using for years. As their five-year contract with the telephone company is up for renewal, they wish to replace the expensive hosted service with an in-house solution.

They will be contracting with us to provide the deployment, ongoing support, and maintenance of their new phone system.

ACMEsoft employs four first-tier support engineers, two second-tier support personnel, and one third-tier support specialist. Each member of each tier has similar talents and can handle the same calls.

They employ a receptionist, an operator, and an administrative assistant. There are 20 programmers, five testers, four project managers, and one person in the shipping department.

The Discussion

Asterisk is an appropriate choice for ACMEsoft. Asterisk provides all of the features common to Centrex solutions, and then some. Since they have no illusions of having an in-house tech to administer Asterisk, only our knowledge set is in question. Since we are professionals who specialize in using Asterisk, we will be able to make it work according to ACMEsoft's expectations.

Asterisk is a powerful alternative to the more expensive hosted solutions. When using Centrex service, each extension must have an analog line. These lines are expensive to install, move, and maintain.

With the current usage statistics, a Primary Rate ISDN (PRI) line makes the most sense. The reason for this is that we will need less than 23 concurrent lines. PRI allows us to use advanced signaling; also, echo is less likely with a PRI than with POTS lines. PRI is often cheaper than having 23 separate POTS lines coming in to our server. Therefore, for this installation, PRI makes the most sense.

With Centrex service, each extension usually gets a unique phone number so that it may be reached from the outside world. To have the same feature, we will be using Directed Inward Dialing (DID) numbers. Usually purchased in blocks of 20, each number can be mapped to an extension, group of extensions, or a service, such as conferencing or voicemail. These numbers are generally inexpensive.

In this example, we will assume the phone company will be providing the full 10-digit phone number for each phone call. This is a very common configuration, which should be available from any phone company. We should always request the full 10 digits in case we have the same last four digits for two telephone numbers coming into our system.

For our connection to the PSTN, we will be using Digium's T100P. This T1 card supports ISDN signaling and integrates well with Asterisk. For our handsets, we will be using Polycom's SoundPoint IP300s using SIP.

The support personnel will be organized into queues; each level of support will have one queue. The operator will also have a queue, as he often receives multiple calls simultaneously.

The Configuration

These are the configuration files for ACMEsoft's PBX. These files assume we have already set up our server as previously discussed.

zaptel.conf

```
#incoming PRI 1
span=1,1,0,esf,b8zs
bchan=1-23
dchan=24
loadzone = us
defaultzone=us
```

We are using ESF framing and B8ZS coding. These are very common in the United States for PRIs.

zapata.conf

```
switchtype=national
context=incoming
signalling=pri_cpe
group=1
channel => 1-23
```

Here we are setting channels 1 through 23 (the channels that take actual calls; channel 24 is for signaling) to be in group 1, and we are telling incoming calls to go to the context called `incoming` in the dialplan.

musiconhold.conf

```
[classes]
default => quietmp3:/var/spool/asterisk/defaultMOH,-z
```

Here we have a general Music on Hold instance, called `default`.

agents.conf

```
[agents]
ackcall=yes
wrapuptime=0
musiconhold => default
updatecdr=yes
;Tier 1
group=1
agent => 1111,0596,John Smith
agent => 1209,0522,William Krandal
agent => 0186,1129,Lindsey Cramer
agent => 0416,0106,Stephanie Lewis
;Tier 2
group=2
agent => 2345234,3489,Likes Longnum
agent => 5692,4989,Smitty Rodriguez
;Tier 3
group=3
agent => 1,1,Forgets Ownname
;Operator
group=4
agent =>0,1234,Operator Console
```

Notice that we can have variable agent IDs. This is usually not a very good idea, as having consistent lengths for IDs is easier to support; however, often politics will dictate whether the length can be standardized.

queues.conf

```
[general]

[default]

;
;Tier 1 Support Queue
[Q110]
music=default
strategy=leastrecent
maxlen=0
context=default
member => Agent/@1

;
;Tier 2 Support Queue
[Q120]
music=default
strategy=ringall
maxlen=0
context=default
member => Agent/@2
```

```
;
;Tier 3
[Q130]
music=default
strategy=leastrecent
maxlen=0
context=default
member => Agent/@3

;
;Operator Queue
[Q100]
music=default
strategy=ringall
maxlen=0
context=default
member => Agent/@4
```

Notice that each queue has its own section. We have configured each queue to have no limit as to length. We will be using some nifty options in the `extensions.conf` file to limit how long callers will be on hold, as setting the options upon entrance seems to be more reliable than setting them in the `queues.conf` file.

sip.conf

```
[general]
context=default
port=5060
bindaddr=0.0.0.0
disallow=all
allow=ulaw

[101]
type=friend
context=local
callerid=ACMEsoft Operator<555-555-1234>
host=dynamic
secret=mypass101
dtmfmode=inband
mailbox=101

[102]
type=friend
context=longdistance
callerid=Sharon Stone<555-555-1235>
host=dynamic
secret=mypass102
dtmfmode=inband
mailbox=102

[111]
type=friend
context=default
callerid=John Smith<111>
host=dynamic
secret=mypass111
dtmfmode=inband
mailbox=111

. . .
```

As you can see, a clear pattern is emerging in this file. We simply copy and paste these configurations to create all 40 extensions needed. Since we have all matching phones, we know that the DTMF mode will be the same for all of them. Also, since we are providing voicemail to all of our users, that will also be similar from user to user.

We should also take care to put our users in the proper context. Our first-level support agent can only call internal extensions; our operator can dial local and toll-free numbers, and our administrative assistant can dial long distance.

The rest of this example assumes we have created the rest of the necessary entries; for the sake of brevity, they have been omitted here.

meetme.conf

```
[rooms]
conf => 850
conf => 851
conf => 852
conf => 853
conf => 854
conf => 855
conf => 856
conf => 857
conf => 858
conf => 859
```

Here we have created 10 conference rooms, with no passwords assigned.

voicemail.conf

```
[general]
format=wav49|gsm|wav
serveremail=asterisk@mydomain.com
attach=yes
maxmessage=180
minmessage=3
maxgreet=60
skipms=3000
maxsilence=10
silencethreshold=128
maxlogins=1
fromstring=The Greatest PBX IN THE WORLD!!!

[zonemessages]
eastern=America/New_York|'vm-received' Q 'digits/at' IMp
central=America/Chicago|'vm-received' Q 'digits/at' Imp

[default]
100 => 100,Operator Queue Mailbox,,tz=central
101 => 123,Operators Mailbox,,tz=central
102 => 674,Patty Smalley,,tz=central
111 => 38594,John Smith,,tz=eastern   ;Support Department works by
                                      ; ETZ
112 => 65413,William Krandal,,tz=eastern
113 => 654,Lindsey Cramer,,tz=eastern
114 => 0106,Stephanie Lewis,,tz=eastern
. . .
```

As we can see, configuring voicemail is simple. The important thing to remember is that whatever we set the name to determines whether an extension will match an entry in the Directory. Also, the context in voicemail should always match the context in `extensions.conf`.

extensions.conf

```
[general]
static=yes
writeprotect=no

#include macros.incl
#include incoming.incl
#include outgoing.incl
#include default.incl
#include dialext.incl

[globals]
TRUNK=Zap/g1
TRUNKMSD=1
```

This is our entire `extensions.conf` file. By using the `#include` feature, we are able to make our configuration files much easier to read, and much easier to maintain. We should remember to keep the filenames easy to read and logical. Since each of these files is included into the `extensions.conf` file, they will not get separate sections in this chapter.

```
;macros.incl
;#included into extensions.conf
[macro-stdexten]
;
; Standard extension macro:
;   ${ARG1} - Extension (we could have used ${MACRO_EXTEN} here as
;                        well)
;   ${ARG2} - Device(s) to ring
;
exten => s,1,Dial(${ARG2},20)        ; Ring the interface, 20 seconds
                                     ;  maximum
exten => s,2,Goto(s-${DIALSTATUS},1) ; Jump based on status

exten => s-NOANSWER,1,Voicemail(u${ARG1}) ; If unavailable, send to
                                          ;  voicemail
exten => s-NOANSWER,2,Goto(default,0,1)   ; If they press #, go to
                                          ;  Operator

exten => s-BUSY,1,Voicemail(b${ARG1})  ; If busy, send to voicemail
                                       ;  with busy message
exten => s-BUSY,2,Goto(default,0,1)    ; If they press #, go to
                                       ;  Operator

exten => s-CHANUNAVAIL,1,Voicemail(u${ARG1})
exten => s-CHANUNAVAIL,2,Goto(default,0,1)

exten => s-.,1,Goto(s-NOANSWER,1)     ; Treat anything else as no
                                      ;  answer
```

```
exten => a,1,VoicemailMain(${ARG1})    ; If they press *, send to
                                       ;  VoicemailMain

[macro-novm]
exten => s,1,Dial(${ARG1},30)      ;ring the interface for 30 seconds
exten => s,2,Goto(default,s,1)
exten => s,102,Goto(default,s,1)
```

Notice that we have a macro to set up all of the extensions we will be creating. This will save us a ton of work later on, as well as make our configuration files very readable.

```
;incoming.incl
;#included from extensions.conf
[incoming]
exten => 5555551234,1,Goto(default,100,1) ;Main number rings to
                                          ;  Operators
exten => 5555552345,1,Goto(default,110,1) ;Direct number to Support
exten => 5551110001,1,Goto(default,111,1) ;Direct line to
                                          ;  Extension 111
exten => 5551110002,1,Goto(default,112,1) ;Direct line to
                                          ;  Extension 112
exten => 5551110003,1,Goto(default,113,1) ;Direct line to
                                          ;  Extension 113
. . .
exten => s,1,Goto(default,100,1);
exten => t,1,Goto(default,100,1);
exten => i,1,Goto(default,100,1);
```

Notice that we handle all incoming calls via this file. Here we define our DIDs and where we want them to ring. We also make sure to create intelligent rules in case the DID information is mangled by our phone company before Asterisk can decode it. In this case, we are sending the calls to our Operator.

```
; outgoing.incl
;#included from extensions.conf
[local]
ignorepat => 9
exten => _9NXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91800XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91866XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91877XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91888XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => default

[longdistance]
ignorepad => 9
exten => _91NXXNXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => local

[trunkdial]
exten => _9.,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})
exten => _9.,2,Congestion(5)
exten => _9.,3,Hangup
```

Notice what we have done here: we created a general context called `trunkdial`, which we use to dial any calls going over the trunk lines. Why is this helpful? If we were to add a new trunk group, we could add only one line. If we were to use the standard method of having each line above dial, we would have to add six lines for each new trunk group.

This example assumes we will have no users placed directly in the `trunkdial` context, such as in the `sip.conf` file. For security reasons, we must be careful that we do not ever place a user explicitly in the `trunkdial` context.

```
;default.incl
;#included in extensions.conf
[default]
exten => s,1,Goto(default,100,1)
exten => t,1,Goto(default,100,1)
exten => i,1,Goto(default,100,1)

; Operator queue, Operator Console, and Receptionist Phone
exten => 100,1,Answer
exten => 100,2,Queue(Q100||||240)  ; only allow 4 minutes in queue
exten => 100,3,Voicemail(u100)     ;  then send to VM
exten => _10[12],1,Macro(stdexten,${EXTEN},SIP/${EXTEN})

;Support Tier 1
exten => 110,1,Answer
exten => 110,2,Queue(Q110||||240)  ; allow 4 minutes in queue
exten => 110,3,Goto(default,100,1) ;  then send to Operator
exten => _11[1-4],1,Macro(stdexten,${EXTEN},SIP/${EXTEN})


;Support Tier 2
exten => 120,1,Answer
exten => 120,2,Queue(Q120||||240)  ; allow 4 minutes in queue
exten => 120,3,Goto(default,100,1) ;  then send to Operator
exten => _12[12],1,Macro(stdexten,${EXTEN},SIP/${EXTEN})

;Support Tier 3
exten => 130,1,Answer
exten => 130,2,Queue(Q130||||240)  ; allow 4 minutes in queue
exten => 130,3,Goto(default,100,1) ;  then send to Operator
exten => 131,1,Macro(stdexten,${EXTEN},SIP/${EXTEN})

;Programmers, extensions 200-219
exten => _2[01]X,1,Macro(stdexten,${EXTEN},SIP/${EXTEN})

;Testers, extensions 251-255
exten => _25[1-5],1,Macro(stdexten,${EXTEN},SIP/${EXTEN})

;Project Managers, exts 301-304
exten => _30[1-4],1,Macro(stdexten,${EXTEN},SIP/${EXTEN})

;Shipping Department, ext 191, doesn't need voicemail
exten => 191,1,Macro(novm,SIP/${EXTEN})

exten => 800,1,Answer
exten => 800,2,VoicemailMain

exten => _85X,1,Answer
exten => _85X,2,MeetMe(${EXTEN})

exten => 888,1,Goto(dialext,s,1)
```

Notice that we are able to create 20 extensions for our programmers in a single line. This is the power of Asterisk's pattern matching, coupled with the flexibility of macros. We can use tricks like this one by grouping similar extensions together.

```
;dialext.incl
;#included from extensions.conf
[dialext]
include => default
exten => s,1,Answer
exten => s,2,DigitTimeout(5)
exten => s,3,ResponseTimeout(20)
exten => s,4,Background(pleaseenterextension) ; "Please enter the
                      ;  extension of the party you are calling."

exten => 9,1,Directory(default)          ; press 9 for the directory
exten => 9,2,Goto(dialext,9,1)

exten => 0,1,Goto(default,100,1)         ; send to operator as a
                                         ;  courtesy if they press 0

exten => i,1,Playback(privacy-incorrect)
exten => i,2,Goto(dialext,s,1)

exten => t,1,Goto(dialext,i,1)
```

This small context allows users to dial anybody in the company, and also to access the corporate directory. The directory, which reads the `voicemail.conf` file, allows access to any extension in the company. By so doing, a "backdoor" line can be established that points directly to this extension, allowing us to no longer need direct phone numbers for each extension.

Conclusions

Asterisk has proved itself again as a powerful solution for real-world problems. By taking advantage of the feature set Asterisk provides, we are able to create a server that has the features of a Centrex system, and then some. The savings ACMEsoft will experience are very real and will pay for the system with a short ROI.

Hosted PBX

Asterisk is not limited to being able to service only one company. With a little finesse, we can configure Asterisk to handle multiple companies without any needing to be aware of the others' presence. Although our example will deal with multiple companies on one site, there is no reason the same principles could not be applied over a high-speed data network.

The Scenario

Al's Computer Depot was a very large computer retailer in the early 1990s, back when computers were fun and profitable. Unfortunately, Al got a bit too used to very high margins on computer sales, and so has moved out of the computer selling business. He and his team have moved into consulting. As a consulting firm, 90% of the employees are traveling at any time.

Al's wife operates a small boutique selling Asian knock-off wallets. Since most of the offices are empty all of the time, Al decided to let her have an office to run her business from. And with such a business, Sue needed a telephone line but it would be no good to have the line answered by someone from Al's Computer Depot (the name wasn't changed as Al didn't want to buy new stationary when he went into consulting).

After this experience, Al decided to sublease more offices, requiring the tenants to purchase telephone service from him. And so, Al's phone system is configured to allow multiple businesses on the same server.

We will be considering three separate businesses: Al's Computer Depot, Sue's Collectibles, and AutoAuction Listings.

The Discussion

Asterisk is perfect for this scenario. The flexible feature set will allow enough features for potential tenants, while being able to be scaled down for the smaller businesses who only need one line, like Sue's Collectibles.

We will be using SIP hardphones and a PRI line for PSTN interconnection. This will give us flexibility to change quickly when tenants come and go.

The Configuration

Once again assuming we have properly installed Asterisk, the following files will configure our server for Al.

zaptel.conf

```
#incoming PRI 1
span=1,1,0,esf,b8zs
bchan=1-23
dchan=24
loadzone = us
defaultzone=us
```

Again, we use ESF framing and B8ZS coding, as they are very common for PRIs in the United States.

zapata.conf

```
switchtype=national
context=incoming
signalling=pri_cpe
group=1
channel => 1-23
```

Here we are setting channels 1 through 23 (the channels that take actual calls; channel 24 is for signaling) to be in group 1, and we are telling incoming calls to go to the context called `incoming` in the dialplan.

musiconhold.conf

```
[classes]
default => quietmp3:/var/spool/asterisk/defaultMOH,-z
```

Here we have a general Music on Hold instance, called `default`.

sip.conf

```
[general]
context=default
port=5060
bindaddr=0.0.0.0
disallow=all
allow=ulaw

[al100]
type=friend
context=al-ld
callerid=Al Getrich<800-555-1234>
host=dynamic
secret=badpassword
dtmfmode=inband
mailbox=100@al
accountcode=al
. . .
[sue1]
type=friend
context=sue-ld
callerid=Sue Getrich<555-555-5555>
host=dynamic
secret=anotherbadpassword
dtmfmode=inband
mailbox=1@sue
accountcode=sue

[aa100]
type=friend
context=aa-ld
callerid=Auto Auctions<555-777-1234>
host=dynamic
secret=1234
dtmfmode=inband
mailbox=100@aa
accountcode=aa
. . .
```

Here we see three of the many SIP extensions that are defined. Notice that two of the users, namely `aai100` and `al100`, both have mailboxes of 100, and both will be extension 100. Since they are in different contexts, though, this will not be a problem.

Also, we should be sure to put each of the SIP users into the correct account codes. By so doing, we can correctly bill calls made to the party who made them. There are a whole host of billing solutions available for Asterisk; however, we focus on what can be done with the default setup here.

voicemail.conf

```
[general]
format=wav49|gsm|wav
serveremail=asterisk@mydomain.com
attach=yes
maxmessage=180
minmessage=3
maxgreet=60
skipms=3000
maxsilence=10
silencethreshold=128
maxlogins=1
fromstring=The Greatest PBX IN THE WORLD!!!

[zonemessages]
central=America/Chicago|'vm-received' Q 'digits/at' Imp

[al]
100 => 100,Al Getrich,,tz=central
. . .

[sue]
1 => 1,Sue Getrich,,tz=central

[aa]
100 => 1234,Auto Auctions,,tz=central
. . .
```

This is a sample of how the `voicemail.conf` file will look, with one sample from each company. As we can see, configuring voicemail is simple. The important thing to remember is that whatever we set the name to determines when an extension will match an entry in the directory. Also, the context in voicemail should always match the context in `extensions.conf`. This allows each company to have its own directory, if it so chooses.

extensions.conf

```
[general]
static=yes
writeprotect=no

#include macros.incl
#include al.incl
#include sue.incl
#include aa.incl
#include outgoing.incl
```

```
[globals]
TRUNK=Zap/g1
TRUNKMSD=1
```

This is our entire `extensions.conf` file. By using the `#include` feature, we are able to make our configuration files much easier to read and maintain. We should remember to keep the filenames easy to read and logical. Since each of these files is included into the `extensions.conf` file, they are not given separate sections in this chapter.

```
;macros.incl
;#included into extensions.conf
[macro-stdexten]
;
; Standard extension macro:
;   ${ARG1} - Extension (we could have used ${MACRO_EXTEN} here as
;                        well)
;   ${ARG2} - Device(s) to ring
;
exten => s,1,Dial(${ARG2},20)        ; Ring the interface, 20 seconds
                                     ;  maximum
exten => s,2,Goto(s-${DIALSTATUS},1) ; Jump based on status

exten => s-NOANSWER,1,Voicemail(u${ARG1}); If unavailable, send to
                                         ;  voicemail
exten => s-NOANSWER,2,Goto(default,0,1)  ; If they press #, go to
                                         ;  Operator

exten => s-BUSY,1,Voicemail(b${ARG1})    ; If busy, send to
                                         ;  voicemail with busy
exten => s-BUSY,2,Goto(default,0,1)      ; If they press #, go to
                                         ;  Operator

exten => s-CHANUNAVAIL,1,Voicemail(u${ARG1})
exten => s-CHANUNAVAIL,2,Goto(default,0,1)

exten => s-.,1,Goto(s-NOANSWER,1)        ; Treat anything else as
                                         ;  no answer

exten => a,1,VoicemailMain(${ARG1})      ; If they press *, send to
                                         ;  VoicemailMain

[macro-novm]
exten => s,1,Dial(${ARG1},30)            ; ring the interface for 30
                                         ;  seconds
exten => s,2,Goto(default,s,1)
exten => s,102,Goto(default,s,1)
```

We can actually reuse these macros from the previous example; this is why macros are so powerful. By defining things generically enough, we are able to reuse the same configuration in many different scenarios.

```
;al.incl
;#included from extensions.conf
[al]
exten => 8005551234,1,Goto(al,100,1) ;AL's direct number
exten => 100,1,Macro(stdexten,100@al,SIP/al100) ;AL's extension
. . .
```

Notice that we can handle incoming calls and internally dialed extensions. Here we define our DIDs and where we want them to ring. We also define failover behavior in case of bad information from our phone company. Finally, we define the extensions for Al's own business directly in this file.

One somewhat interesting side effect of this method is that if we dial the full 10-digit number from a telephone, it will route it internally, instead of hopping off to the PSTN and then coming back in. Of course, if we dial a 9, then we will still use the trunk rules.

```
;sue.incl
;#included from extensions.conf
[sue]
exten => 5555555555,1,Goto(sue,1,1) ; Sue's direct number
exten => 1,1,Macro(stdexten,100@sue,SIP/sue1) ; Sue's extension
. . .

;aa.incl
;$included from extensions.conf
[aa]
exten => 8005551234,1,Goto(al,100,1) ; Only phone number for A A
exten => 100,1,Macro(stdexten,100@aa,SIP/al100) ; Only AA extension
```

Here we have the other two businesses. We have chosen to configure them much the same way as we did for Al. Each business will be in its own configuration file.

```
;outgoing.incl
;#included from extensions.conf
[al-local]
ignorepat => 9
exten => _9NXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91800XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91866XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91877XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91888XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => al

[al-ld]
ignorepad => 9
exten => _91NXXNXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => al-local

[sue-local]
ignorepat => 9
exten => _9NXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91800XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91866XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91877XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91888XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => sue

[sue=ld]
ignorepad => 9
exten => _91NXXNXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => sue-local
```

```
[aa-local]
ignorepat => 9
exten => _9NXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91800XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91866XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91877XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
exten => _91888XXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => aa

[aa-ld]
ignorepad => 9
exten => _91NXXNXXXXXX,1,Goto(trunkdial,${EXTEN},1)
include => aa-local

[trunkdial]
exten => _9.,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})
exten => _9.,2,Congestion(5)
exten => _9.,3,Hangup
```

Here we used the same `trunkdial` context as in the previous example for exactly the same reasons. However, with all these different outgoing contexts, the complexity of adding extensions would simply keep increasing.

Each company has a unique set of outgoing contexts so that only its extensions are included. This helps ensure the correct extension is reached when extensions exist in more than one context, such as extension 100 existing in contexts `aa` and `al`.

Compared to our previous example, the system offers fewer features for our users: we do not have conferences, we cannot get to the directory, and we must call our own extension and press the * key to get to our voicemail. However, these features can be activated at will, or even as customers pay to have them added.

Conclusions

Asterisk has again been used to fulfill a different need in a phone system. By taking advantage of contexts, we have been able to create multiple virtual phone servers with only one server, one PRI line, and one set of configuration files.

Summary

In looking at these case studies, I hope you have been able to spot common features in what we are trying to implement in each different situation. We can learn from these implementations and apply many of the same strategies when we encounter users with similar needs.

[PACKT PUBLISHING]

9
Maintenance and Security

Now that we have an Asterisk server installed and running we should consider the maintenance and security of the server. There are a number of aspects involved here and we will cover each in turn. Since the Asterisk server is going to be the central hub of our phone system, the importance of securing and maintaining it is obvious, as without it we lose our primary means of communication.

This chapter also looks at scalability issues, which are important to keep Asterisk running at high loads. Finally, the last section takes a look at how to get support from the community, and how to stay abreast of Asterisk developments. Keeping up with what's going on in the Asterisk world can be a very useful way to stay prepared for potential problems before they happen, as well as providing a helping hand should things go wrong.

Backup and System Maintenance

One of the most important aspects of maintaining a system is the update or patch management process. It's vital that we keep our system up to date in order to reduce bugs and ensure that any security vulnerabilities in our software are fixed as soon as possible. When updating our Asterisk server, there are three main areas to maintain:

- The Asterisk service itself
- The various components that Asterisk depends on (Zaptel, libpri, festival, etc.)
- The host OS and any supplementary tools installed (OpenSSH, mpg123, etc.)

When we discussed setting up the Asterisk server, we covered installing from the CVS source repository, which is an easy way to keep up to date, as you can continually download the latest version.

If we were to manage all applications as source packages only, we would need to go through the steps given in the installation chapter for each component as updates are released. However, when we factor in the updates for the tools of the host operating system and any other tools we use, this can become tedious and error prone quite quickly as these extras aren't contained in the Asterisk CVS repository. It is at this point we should consider a package management system to ensure we keep everything up to date automatically, reducing our administrative burden. The options we have for this are dependent on the Linux distribution we decided to use.

Examples are:

- APT: used by Debian (and ported to many other distributions such as Red Hat). See `http://www.debian.org/doc/manuals/apt-howto/index.en.html`.
- Portage: Gentoo. `http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2&chap=1`.
- URPMI: used by Mandrake. See `http://www.urpmi.org`.
- Yum used by YellowDog/CentOS. See `http://linux.duke.edu/projects/yum/`.

Each of these is documented on its respective site. In order to ensure our system remains well maintained, we must become familiar with the package management tools at our disposal.

Backing Up Configurations

In the installation chapter, we briefly touched on backups and mentioned making a copy of the configuration files before editing them. This is good administrative practice and will protect us from any mistakes we make in the configuration files. However, it will not protect our configuration if the system were to be compromised or if we were to lose the media containing the configuration. It also doesn't take into account any data the system holds that we might lose.

Asterisk configurations can become quite involved over time and we invest a lot of time in setting these up. Repeating this work would be far from desirable in the event of system failure. For this reason we should keep off-line copies of the configuration files.

The backup method is unimportant; what is important is that we understand it and are sure that it fits our need; in the end, the final choice will mostly be down to personal preference. Above we have chosen some of the most common methods to provide practical examples; we can choose any backup solution that suits us or fits with our business requirements as long as we ensure that we do backup—**configuration files, data, and logs.** The locations of these files are configurable and may change depending on our distribution. You can find their locations in the `/etc/asterisk/asterisk.conf` file:

```
[global]
astetcdir => /etc/asterisk
astmoddir => /usr/lib/asterisk/modules
astvarlibdir => /var/lib/asterisk
astagidir => /usr/share/asterisk/agi-bin
astspooldir => /var/spool/asterisk
astrundir => /var/run/asterisk
astlogdir => /var/log/asterisk
```

A simple manual copy of the `/etc/asterisk` directory may seem to be enough to ensure we don't lose these settings but we have to consider files outside the Asterisk directory that the server directly relies on, such as `/etc/zaptel.conf` and our network configuration files. The locations of some of the files important to us can vary with distribution, but invariably they are kept as a sub-folder of the `/etc/` directory. This means that as long as we ensure that this directory is copied from the server periodically we can restore the configuration files in the event of a failure.

We should ensure this is done automatically at least once per week. There are a number of options for doing so but the simplest and easiest to automate is an `rsync` copy to our backup server, or if the Asterisk server has a tape drive we could use the `tar` utility.

Backup Schedule

Our backup schedule may have to be more frequent than once per week depending on business and personnel requirements.

Below is a sample `rsyncd.conf` configuration file:

```
[asterisk_backup]
   path = /home/adminuser/asterisk_backups
   comment = Asterisk Backups
   uid = nobody
   gid = nobody
   read only = no
   list = no
   auth users = username
   secrets file = /etc/rsyncd/asterisk
```

The `/etc/rsyncd/asterisk` file would contain our username and password pair; this and the `/etc/rsyncd.conf` file would be on the backup server, which we would also install and run `rsyncd` on.

We could then run the following command on the Asterisk box to back up the configurations:

```
$ rsync --verbose --compress --progress --recursive --times --perms
   --nodelete /etc/* backupserver:asterisk_backup
```

This would copy everything from the `/etc/` directory on the Asterisk server to the `/home/adminuser/asterisk_backups` directory on the backup server.

Added Security

If we install SSH on the backup server, we could perform the above over an encrypted SSH tunnel, by altering the command as follows:

```
$ rsync --rsh=/usr/local/bin/ssh --verbose --compress –progress
   --recursive --times --perms --nodelete /etc/*
   backupserver:asterisk_backup
```

We could also back up to a tape drive attached to the Asterisk server by using the `tar` utility, which requires no configuration files as it is completely command-line driven:

```
$ tar --verbose -j --create /etc/*
```

The `-j` option compresses using `bzip2` for efficient storage.

If we want to use an archive file instead of a tape drive, we add the `--file` (or `-f`) parameter:

```
$ tar --file=asterisk_backup.tar.bz2 --verbose -j --create /etc/*
```

We could automate these processes by adding the relevant command lines to cron. For example:

```
$ crontab -e
 0 2 * * 0 tar --file=asterisk_backup.tar.bz2 --verbose –j
           --create /etc/*
```

This would back up the configuration files at 02:00 every Sunday morning. It may be worth studying cron if we are unfamiliar with it to ensure we understand its capabilities.

Backing Up Voice Data

We would also want to ensure we had backups of the saved data that we have in our spool directory. This is where our hold music and voicemail is located for example. We would almost certainly want to restore this in the event of a system failure, especially when we have referenced audio files in our configuration files and so they are required for Asterisk to function properly. The default location of the spool directory is `/var/spool/asterisk`, so we should add this to our backup commands above.

Using `rsync`:

```
$ rsync --verbose --compress --progress --recursive --times –perms
   --nodelete /etc/* /var/spool/asterisk backupserver:asterisk_backup
```

Using `tar`:

```
$ tar --file=asterisk_backup.tar.bz2 --verbose -j --create /etc/*
   /var/spool/asterisk
```

This would ensure our backups contain all our configuration and voice data necessary for a complete restoration of the Asterisk server.

Additional Considerations

We may also want to back up other data and log files if we have additional applications installed with the Asterisk server, such as Webmin.

Backing Up Log Files

It is important to ensure that any logs we have are backed up too, and we can do this exactly as above by adding `/var/log/asterisk` to the commands. It may not always be necessary to keep log files; however, we may have to keep these if we have a data retention policy or are under regulations that require log and data to be kept.

Policies

If we are under a regulation or policy such as Sarbanes-Oxley or HIPAA, then we will have clearly defined rules for log retention and may have to ensure compliance of the Asterisk backups.

Backup Scripts

In order to back up the system effectively it's important that we back up the system incrementally so that we can go back to previous points in time e.g. before a disaster happened or before a configuration change that caused problems. The scripts that follow allow is to do so by keeping copies of our configurations and data on a backup server, and running these daily means we can go back and restore any date's configuration if necessary.

The following scripts allow us to backup our files to a back up server running an SSH daemon. They can be used as is (with the obvious change to the `backup_server` variable) or as a starting point for a backup system. The files are commented enough to make them self-explanatory. They use a combination of `tar` and `scp` to archive the files and copy them off the server. Alternatively, you could use `rsync` by following the guidelines shown earlier.

First, here is the `backup.cron` script:

```
#!/bin/bash

########################################
## Backup script for asterisk       ##
########################################
## This script is designed to make  ##
## a copy of all important config   ##
## and data files, which will then  ##
## be copied to a backup server     ##
## running an ssh daemon            ##
########################################
## Usage: backup.cron, no           ##
## arguments required.              ##
########################################

# edit variable below to contain your
# backup server user and hostname as
# well as directory location
backup_server="username@backupserver:/path/to/backups"
date='date +%Y-%m-%d'

# Remove old backups to keep from filling the disk with junk
rm /backup/*.tar.gz -f

# Backup the /etc/ directory
tar cfz /backup/asterisk-configs-${date}.tar.gz /etc

# Backup the voicemail directory
tar cfz /backup/asterisk-vm-${date}.tar.gz
/var/spool/asterisk/voicemail

# Rotate the logs for Asterisk
/usr/sbin/asterisk -rx 'logger rotate'

mv /var/log/asterisk/debug.0 /tmp/debug.${date}
mv /var/log/asterisk/messages.0 /tmp/messages.${date}
mv /var/log/asterisk/event_log.0 /tmp/event_log.${date}

# Backup log files
tar cfz /backup/asterisk-astlogs-$date.tar.gz /tmp/*.${date}

# Remove unnecessary files
rm -f /tmp/*.${date}

# Copy all archives to our backup server.
scp -B /backup/*-configs-${date}.tar.gz $backup_server
scp -B /backup/*-vm-${date}.tar.gz $backup_server
scp -B /backup/asterisk-astlogs-* $backup_server
scp -B /backup/asterisk-phonecfg-* $backup_server
```

Next, let's see `monitor_mix.cron`:

```
#!/bin/bash

################################################
## Shell script to handle phone monitoring    ##
## files.  This script will be run at night   ##
## and maybe at lunch.  It will use soxmix    ##
## to mix the in and out components of the    ##
## conversation, delete the in and out com-   ##
## ponents, and then use lame to encode the   ##
## mixed wav file into an MP3                 ##
################################################
## Usage: monitor_mix.cron, no arguments      ##
## required                                   ##
################################################

# edit variable below to contain your
# backup server user and hostname as
# well as directory location
backup_server="username@backupserver:/path/to/backups"
date=`date +%Y-%m-%d`

# Clear previous backup files prepare folder
# structure for backup set.
rm /backup/monitor -rf
mkdir /backup/monitor
mkdir /backup/monitor/${date}
chmod -R 700 /backup/monitor

# For each conversation in the monitor directory
# soxmix the two parts of the conversation
# together and convert to mp3.
cd /var/spool/asterisk/monitor
for i in 'ls *-in.wav'
do
  basename='basename $i -in.wav'
  echo $basename

  # soxmix "in" and "out" files
  soxmix $i $basename-out.wav $basename.wav
  rm -f $i
  rm -f $basename-out.wav

  # convert resulting wav to mp3
  lame --resample 16 -m m -b 32 -h --cbr $basename.wav $basename.mp3

  # Remove unnecessary files
  rm -f $i
  rm -f $basename-out.wav
  rm -f $basename.wav

done

# Put newly created mp3s in local backup directory
mv *.mp3 /backup/monitor/${date}

# Copy mp3s to our backup server running an sshd
scp -B -r /backup/monitor/* $backup_server
```

We could also restart the Asterisk service nightly in order to resolve any hung channels; this may not be possible if the system sees high usage during the night, but is useful in situations where we have channels hanging often.

Finally, `asterisk_restart.cron`:

```
#!/bin/bash
/etc/init.d/asterisk stop
/etc/init.d/zaptel restart
/etc/init.d/asterisk start
```

Time Synchronization

As our Asterisk system retains logs of calls and also makes routing decisions based on time, it is important to have the system clock synchronized. We can do this with Network Time Protocol (NTP). This is very easy to use. Just install the `ntpdate` program, which you will most likely find in your distribution's package management system (`yum`, `urpmi`, `apt`, or whatever). Then run the script shown below. If we have a local time server (for instance if we have other time-dependent services such as Kerberos authentication installed), we should use that.

The `timesync.cron` NTP script:

```
#!/bin/bash
ntpdate pool.ntp.org # replace pool.ntp.org with local time server
/sbin/hwclock –-systohc # sync the hardware clock
```

Adding It All to cron

The four files we have created can be added to our crontab to ensure they are run periodically. We can do this by first running:

```
$ crontab –e
```

Then creating the following entries (replacing `/path/to` with the location of our scripts):

```
30 01 * * *            /path/to/backup.cron
59 23 * * *            /path/to/monitor_mix.cron
00 01 * * *            /path/to/asterisk_restart.cron
00,15,30,45 * * * *    /path/to/timesync.cron
```

This ensures that our `backup.cron`, `monitor_mix.cron`, and `asterisk_restart.cron` scripts are run nightly and that our time is synchronized every 15 minutes.

As we are running these commands non-interactively, i.e. we are using `scp` in batch mode, we must ensure that the `scp` command can authenticate with the backup server. To do this we should use SSH keys instead of passwords for authentication by running the following commands:

```
$ ssh-keygen -t dsa ; accept all defaults by pressing enter at each
                    ;  prompt
$ ssh username@backupserver 'cat >> ~/.ssh/authorized_keys'
   < ~/.ssh/id_dsa.pub ; enter password when prompted
```

Rebuilding and Restoring the Asterisk Server

If the unthinkable happens and we lose our server due to hard disk failure or if we have to rebuild because of a system compromise, we need to know exactly how to get the server back online as fast as possible to minimize downtime.

There are a number of steps involved in this:

1. We rebuild the server, by following the instructions in the installation chapter. We follow exactly the same process up to the point of configuration. Since we have a backup of the configuration files, we can skip this part and replace the files with our backups later.
2. We replace the configuration files. We identify the latest usable backup, from which we extract the `/etc/` directory. We then replace the operating system's configuration files we need and replace the `/etc/asterisk` directory. This ensures we have our previous configuration.

   ```
   #!/bin/bash
   $ tar xjvf asterisk_backup.tar.bz2
   $ cp -R etc/asterisk /etc/asterisk
   ```

   We follow a similar process for any other configuration files we wish to restore.
3. Data and Logs. We follow the same process as in Step 2, but this time restoring the `/var/spool/asterisk` and the `/var/log/asterisk` directories to their original locations as required, as well as restoring any other areas of the system we have backed up.
4. Permissions. The last thing we need to ensure is that Asterisk can read and write to the files necessary to function.

At this point we are able to restart the Asterisk server and verify that the system works properly by testing that we can make and receive calls and checking that all features of the system are functioning as they were previously. We could also ensure that no errors appear, after which it can be reintroduced back to its production environment.

Disaster Recovery Plan (DRP)

If during installation we document as much as possible and create a valid DRP, we will be able to get our Asterisk server back online with minimal disruption and effort. Since Asterisk is most likely an essential line of communication to partners and customers, downtime is an extremely important aspect to consider and creating a DRP should be addressed long before any disaster occurs.

Even something simple such as logging the installation process and documenting how to restore from backups is enough for at least a basic DRP, although it is recommended that we go further and create a full plan for not only the Asterisk server but also all other mission-critical services. We could also possibly have a layer of fault tolerance built into the system.

Our plan must take into account at least the following:

- Notification of any outage or data loss
- Times when outage of the Asterisk service would be most detrimental to the organization
- Responsibility for getting the service restored
- Estimated time scale for restoration of service
- Location of backups and other necessary files
- Vendor support contact details
- Detailed restoration instructions, which would include:
    - Restoring the service
    - Restoring all data
    - Restoring all configurations
    - Re-implementing backup procedures

Asterisk Server Security

Before we cover external security and before we think of putting the Asterisk server onto our production network, we must consider the internal security of the system to ensure that it fits with our security policy and meets good security practice at least.

Internal host security can be achieved in a variety of ways and there are many applications and tools that we can use to aid us in this. We will not discuss all of the tools and add-ons we can use for generic system security, but we will, however, cover basic operating system hardening with Asterisk in mind, as well as further steps we can take to ensure that the Asterisk system is running as securely as possible.

It is also important to consider the physical security of the Asterisk system. We may want to have it under lock and key along with our other important infrastructure devices.

Internal Access Control

One of the most important and most overlooked aspects of host-level security is physical security. Our Asterisk server is a communication channel and most likely carries some confidential information. Be sure to have it as segmented from other non-essential non-confidential systems as is reasonably possible.

In any multi-user system, internal discretionary access control lists (DACLs) are essential for security and Linux as an OS has Unix foundations for these control lists. There are a variety of permissions that go far beyond read, write, and execute. However, focusing on these is enough for our purposes and will help us maintain a secure system. As a rule we would have no one but administrators accessing the Asterisk server, because our users operate the system transparently from their telephony devices—either a handset or a software telephone. No direct access to the system is usually required. This is assuming that our Asterisk system is installed on a machine on its own that provides no other services, which is not always the case but is *highly* recommended.

Installing the Asterisk service on a dedicated machine offers the following benefits:

- Resources are dedicated to Asterisk and are therefore easier to monitor:
  - We know if Asterisk requires extra resources.
  - Badly performing services don't affect Asterisk.
- The attack surface of the machine is reduced:
  - There are fewer avenues for remote attack if the Asterisk service is the only way in.
- Maintenance of the system is easier:
  - We don't have to check Asterisk updates for compatibility with other services and vice-versa.
  - A system's reliability and uptime are inversely proportional to the number of services and users it provides for.
  - There won't be downtime of the Asterisk service while unrelated services or components are updated, modified, or removed.

We should ensure that the Asterisk service has access to the directories it requires to perform its function. The permissions mentioned here are not the default but create a more secure setup, and we should test that our Asterisk service functions properly after making these changes. The default permissions on the directories listed are usually **755** and for the files **530**. The directory locations listed below are the default directories and permissions for an Asterisk install on Debian; consult your `asterisk.conf` file to confirm their exact locations on your distribution.

The key directories are:

/etc/asterisk

The configuration files for Asterisk are here. The Asterisk service requires read access to this directory so that it can read its configuration as it loads up and prepares itself for use. It won't need write access, as we will modify these files outside Asterisk and let Asterisk read them as it needs them.

/usr/lib/asterisk/modules

Asterisk has a variety of add-ons and different functionality provided by modules, which are shared libraries that Asterisk can load as needed. They are stored in this directory, and read access is all that's required. We don't require write or execute access as these modules aren't executed directly but loaded by an already running program (the Asterisk binary).

/var/lib/asterisk

This folder contains required files such as public keys for services. Read access is required so that Asterisk can read and present these keys to service providers. Write access is not required as when new keys are added or keys change we would modify them manually.

/usr/share/asterisk/

This contains common files such as sounds. Again only read access required so that Asterisk can load and play the sounds when necessary.

/var/spool/asterisk

This is the spool directory for storage of voicemail messages and other data. Here read/write access is required. Asterisk stores this data in real time and therefore needs to be constantly writing to this directory. We wouldn't manually edit anything here usually, although we may occasionally delete old data.

/var/run/asterisk

Asterisk stores the PID of the currently running service here, and so it requires read/write access.

/var/log/asterisk

Asterisk keeps a variety of log files here and requires read/write access to continually log information and errors relating to the Asterisk service.

The following short script will set the permissions outlined above:

```
#!/bin/bash
# Sets minimum permissions required for Asterisk's key directories
# Modify directory locations based on your /etc/asterisk.conf file

# Make files root owned and asterisk grouped so that a compromise of
# the asterisk user doesn't allow write access to the config files

chown -R root:asterisk /etc/asterisk /usr/lib/asterisk/
 /usr/share/asterisk/ var/spool/asterisk /var/run/asterisk
 /var/log/asterisk

# Give owner (root) r/w/x and give group (asterisk) r/x

chmod 750 /etc/asterisk /usr/lib/asterisk/ /usr/share/asterisk/
 /var/spool/asterisk /var/run/asterisk /var/log/asterisk


#Additional write access for asterisk group on necessary directories

chmod 770 /var/spool/asterisk /var/run/asterisk /var/log/asterisk
```

If you have further Asterisk add-ons installed, the permissions of those files and directories will also likely require modification to increase security.

Host Security Hardening for Asterisk

When we have our basic DACLs in order, we can consider a number of other methods for keeping the Asterisk system secure.

There are several tools that can be installed and used to improve security on Asterisk, and describing the options for many of them would take up entire bookshelves of their own. Here, we will discuss some of the simpler tools for keeping you informed on how secure your system is.

Integrity Checker

We could install Tripwire or another file integrity checker to monitor the checksums (hash values calculated from a file's contents) to ensure that the contents of a file haven't changed. This helps by informing us whenever a file changes; more specifically it focuses on binary files. So if an attacker succeeded in altering the Asterisk binary or one of the modules you would know about it. You can also monitor other operating system files (`netstat`, `ps`, `top`, etc.) to ensure that they haven't been tampered with. The security offered in this is knowing which things have changed without your approval in the event of a system compromise.

Checksums

A checksum is a calculated by running a file's binary contents through a known algorithm, giving a constant value as long as the file contents do not change (the file name has no relevance on the sum). This is used by tools such as Tripwire to determine if a file has changed.

Root-Kit Detection

Root-Kits are tools installed by attackers in order to gain control of the system. They modify binary files, change kernel system calls and use a variety of other evasion techniques such as covert communication channels. This all is in aid of keeping the attacker hidden so that we don't know they have compromised us, leaving them free to plunder the system and use our resources as they see fit. A root kit detection tool is useful as it helps us find these root-kits and quite often helps remove them. The two most notable tools are **rkhunter** and **chkrootkit**.

- `http://www.rootkit.nl/`—rkhunter
- `http://www.chkrootkit.org/`—chkrootkit

Automated Hardening

We can also use a tool such as Bastille, which will help us harden areas of the system outside of Asterisk. We can implement a host-based firewall and modify other system settings to increase security. Bastille has a wizard-based interface, which asks a series of questions of the user and then creates and applies a security policy based on the given answers. It requires very little knowledge of the underlying system and is a generous boost to the overall security of the Asterisk host.

There is a plethora of other tools to choose from; however, these are very common and very easy to use and are almost essential to a secure system. Installing and using these at a minimum provides the knowledge of what's going on within our system, which is an important part of knowing how secure we are.

Role Based Access Control (RBAC)

It's long been known that the traditional DACL, which is prevalent in many OSs including Windows and Linux BSD as well as other Unix-based systems is not the only way nor the best way to separate system access. RBAC is not an entirely new idea, and it has been around for a long time but doesn't see much usage due to being quite difficult to implement.

Asterisk is a very complicated system, which performs a variety of functions, so it can be very difficult to create a workable access control list for access to system resources. RBAC works by not having a single root or administrative user, but instead splitting all tasks to only those users in the system that require them.

RBAC can be provided on Linux by using RSBAC (Rule Set Based Access Control) found at `http://www.rsbac.org`. You can use Adamantix, which is a distribution that has this already fully implemented, and there are configurations of the system available to set up Asterisk.

- `http://www.adamantix.org/`—Adamantix

SELinux

SELinux is a patch for the Linux kernel produced by the NSA for their purposes, and it is described as experimental. It is, however, used on a variety of production networks as an implementation of the Mandatory Access Control theory.

The good news for Asterisk administrators, however, is that there is a pre-written script for SELinux downloadable from `http://www.coker.com.au/selinux/` if you use Debian. The policy is also available for other distributions such as Gentoo, where you will find it within Portage.

Network Security for Asterisk

As many of the protocols Asterisk supports are used over a TCP/IP network, we need an understanding of how to control and firewall these correctly to ensure we only let the necessary traffic pass through.

Our firewall will most likely be on a box separate from our Asterisk installation and placed at the network perimeter (we may also have a host-based firewall to which different rules may apply). In order to define the required rules, I won't detail how to configure a specific firewall product, but provide the details necessary to configure any device we have protecting our Asterisk installation.

These rules would apply to any device, be it `iptables` on a Linux machine, a commercial firewall such as Microsoft ISA server or checkpoint, PIX, etc. The product in use isn't the main issue, the protocol rules that are required are. We can then take these generic rules and apply them to any firewall device we decide to install.

Firewalling the Asterisk Protocols

When it comes to security, firewalls have traditionally been the most important mechanism for protecting internal company assets. For your Asterisk implementation and more specifically the VoIP elements of this system, this is an important consideration.

VoIP protocols are among the most complex in common use and require a great deal of forethought before we can go ahead and deploy Asterisk on our production network. We must consider which solutions we will use and which providers can supply them for us. In order to ensure the network is securely set up, we should have a thorough understanding of the protocols that we'll use so that we can firewall effectively.

It is often difficult to firewall VoIP protocols and there are many extensive documents detailing various scenarios, so here we will discuss the basic needs of a protocol for firewalling.

Probably the two most common protocols used by VoIP communications today are SIP and H.323. At the time of writing, SIP is growing in popularity, although H.323 is the most widespread in production use. The choice of protocols used for our VoIP communications depends entirely on which our vendor supports and which our contacts use.

We will cover SIP, H.323, and IAX here. We have covered these protocols from a technical perspective in previous chapters and we will now see how to firewall these effectively and why IAX is much easier to maintain from a network-control perspective.

SIP (Session Initiation Protocol)

There are now a variety of firewalls that have SIP support built in. All the rules required to allow the protocol through the device are available and all that is needed is for the relevant switch to be flicked. If we have a firewall like this (examples are borderware and ISA server), then our job is done. If, however, we have to define our own firewall rules, a little more work will be required. If we have a traditional firewall, which is a border control mechanism for two networks (usually the LAN and the Internet), then it is relatively straightforward.

We will require:

- Incoming connections on port 5090 (UDP and TCP) to the Asterisk machine in order to receive SIP calls.
- Outgoing connections on port 5090 (UDP and TCP) from the Asterisk server in order to make SIP calls.

If we have Network Address Translation (NAT) between our Asterisk server and the clients accessing it, then things get a little more complicated. In order to get such a setup working, it is suggested that we get a SIP Proxy that supports NAT, which will allow Asterisk to use SIP without difficulty.

The reasoning behind this is that NAT is a hack that was created in order to increase the lifetime of the IPv4 address space. NAT works by taking the internal address (one that isn't Internet routable) and modifying packets sent out so that they use one of the NAT gateway's external addresses. The NAT gateway then takes the returning packets (addressed to it) and rewrites them for the original client based on information held in a lookup table. This works well for most single socket applications (a socket being a pair of IP addresses and ports).

In SIP's case this will not work as well as SIP requires a distinct address for the SIP client and when we use NAT that is obviously not the case, as multiple clients use the same address. The Internet Engineering Task Force (IETF, `www.ietf.org`) maintains the Internet drafts that detail exactly how to get SIP working through a NAT device. The best advice, however, is for us to use a SIP Proxy or attempt to route our SIP service through a router without NAT, that is, to basically give Asterisk one of our publicly accessible IPs. Depending on our placement of the Asterisk server this may be a viable solution. With IAX we can link multiple Asterisk servers so this gives us added options when it comes to server placement.

H.323

H.323 has a similar problem to SIP as it is also designed to require distinct IP addresses and the same advice applies: if we can have the H.323 server on a public IP then it may be easier to maintain, as long as we firewall it effectively. If we have control of both ends of the communication, we can set up a VPN between the two sites, which solves this problem and ensures end-to-end encryption.

To firewall H.323, we need to permit incoming and outbound connections on ports 1720 TCP and UDP ports 5000-5014.

IAX

IAX is a lot more straightforward than either H.323 or SIP as it was designed with the limits imposed by NAT in mind. You can easily allow this traffic through your firewall NAT with minimal fuss.

IAX uses port 4569 UDP outbound and inbound for communication. The old IAX protocol, mentioned in an earlier chapter and succeeded by the current IAX (IAX2), used 5036 UDP.

IAX is also more powerful than either H.323 or SIP and has several features that make VoIP administration and use much easier. For example, it has enhanced signaling capabilities and separates signaling and data more effectively. Also as IAX is not a standard and therefore has no standards body monitoring the decision process, modifications can be made more easily.

RTP—The Real-Time Transport Protocol

RTP is the protocol often used to carry the audio data in a VoIP conversation; it is a standard developed by the IETF (Internet Engineering Task Force). It can also be used to carry video data and is designed specifically to handle this sort of real-time data. It attempts to guarantee that the data will be transmitted and received in a short period of time. Obviously latency in voice conversations can be problem, so RTP avoids this latency as much as possible and concentrates on timely delivery of data.

To allow RTP to function, we would have to allow the following ports inbound and outbound from our Asterisk server: 10000 to 20000 UDP

Controlling Administration of Asterisk

As we have set up Asterisk to access files owned by the root user and Asterisk group, this means that the Asterisk service can read and write only to the files it requires. We, however, may have to perform additional maintenance tasks such as adding extensions, creating new voicemail boxes and so on.

As Asterisk configuration is managed by modifying flat files, we manage this configuration by logging on to the server with an interactive session, at the local console or remotely. To follow best practice, we wouldn't login directly as the root user, but more likely as the Asterisk user. If we did need to edit any files the Asterisk user doesn't have privileges for, then we would switch user to root using the `su` command:

```
$ su -
Password:[root password]
```

We could also implement Sudo and give access to this to our Asterisk user account. Either way would log in as root indirectly. To ensure that no one else can log in as root across the network, we should configure our remote access mechanism to disallow root logins. The most common remote access method for managing an asterisk server would be SSH and the most common implementation OpenSSH `http://www.openssh.com/`, which can be configured to prevent root login by editing the relevant directive in the configuration files:

```
$ cat >> "PermitRootLogin No" >> /etc/ssh/sshd_config
```

We could also further secure our remote access by using the internal firewall (set up earlier by Bastille) to allow access only from the IP address of our administrative team's IP addresses. This would prevent external attackers and internal users from making unauthorized connections to the Asterisk server.

Sudo

Sudo allows us to give restricted administrator access to selected users, but be warned that it is quite easy to misconfigure and give away more access than you intend.

For example, giving someone Sudo access to `vim` gives them the ability to write to all files as root and to execute a root shell from within `vim`. Most likely not desirable!

`http://www.courtesan.com/sudo/`

Asterisk Scalability

As the Asterisk server is most likely highly critical to business, we want to ensure that restoring from backups rarely happens and in the event of losing a machine when an administrator isn't available, we have some sort of fail-over system in place. To achieve this, we apply redundancy and load-balancing techniques to ensure that our infrastructure has the resources to handle the data it needs to process.

In the event of a component failing, we would like to ensure that we don't lose services. Ideally, the users of the system should never know there was any failure and the administrator can get the failed system back online or replaced at the next convenient moment. This sort of forward planning is essential for maintaining a service that will be used as extensively as Asterisk often is.

Take the example of the 24-hour call center. If we have a business that relies on the telephony system in order to generate revenue, then the loss of that service is a loss of revenue. Being a 24-hour service there may not always be an administrator on site—there may be periods where there is only "on call" cover. It would be a waste of resources to have all the users idle while they wait on an administrator possibly being wakened and then ferried to the site in order to get the system back up and running.

As you can appreciate, a single point of failure is not only undesirable but can also have a severe negative impact on the profitability of the business. There is also the chance that if our usage of the Asterisk system outgrows current resources, the Asterisk machine has to be taken offline while it is upgraded. We could avert this by having scalability built into our design from the outset to ensure that the system can grow with business demand.

As Asterisk can't be installed onto a cluster, we require load balancing and scalability that can be implemented without the use of clusters, which isn't as hard as it might seem.

Load Balancing with DNS

One of the most common ways to load-balance a system is to use DNS, the Domain Naming System. This has the ability to "round robin" replies to queries in order to spread load between different machines.

One of the largest DNS load balancing systems that we have all most likely used is the Google search engine.

```
$ dig google.com +short
216.239.37.99
216.239.57.99
216.239.39.99
$ dig google.com +short
216.239.57.99
216.239.39.99
216.239.37.99
$ dig google.com +short
216.239.39.99
216.239.37.99
216.239.57.99
```

As we can see, each time the command to look up the IP of the Google server is run it returns a different IP as the first IP, so that clients accessing it are spread between all of the IPs in the pool and no single machine gets overloaded. You can also add addresses to the pool and remove them without affecting the client, which means this system will scale well to allow many users to access what appears to be just one service.

The advantage of using round robin DNS is that the server hardware behind the service has no direct bearing on how the system can be scaled, as we have the option of adding and removing servers. For instance, were we to suddenly grow we could add in more servers or replace/upgrade existing servers leading to very simple scalability. There is also some redundancy inherent in this system as if your client can't contact the first server it will then attempt to contact the second server. This means that the loss of a server doesn't bring the entire system to a halt, it merely slows it down slightly.

Caching

Since clients cache the IPs they get from DNS, when they find a working IP, the slow down incurred will be negligible. However, it is highly recommended to remove problematic machines or addresses from the pool.

The example we look at here uses A records; however, it is increasingly common to see round-robin implementations for SRV records. SRV records are used to locate a service within a domain. For example in Microsoft's Active Directory implementation, SRV records are used to locate domain controllers and in our Asterisk setup we would use them to locate our routing service providers. The functionality of round robin doesn't differ for the record you request, however—you still create a pool of IPs in your DNS implementation. The two most commonly used implementations of DNS, Microsoft DNS and BIND (versions above 4.9.7), support SRV records and support round-robin SRV records.

For example to set up multiple SRV records for our SIP implementation, we would add the following to our DNS zone:

```
sipa   IN     A      10.1.1.1
sipb   IN     A      10.1.1.2
sipc   IN     A      10.1.1.3

sip    IN CNAME      sipa.example.com
       IN CNAME      sipb.example.com
       IN CNAME      sipc.example.com

_sip._udp.example.com.     IN     SRV    20     0      5060
       sip.example.com.
```

This sets up three SIP servers for us (`sipa`, `sipb`, and `sipc`) and one `_sip._udp` SRV record. Whenever the SRV record is requested, one of these three SIP servers will be returned.

Support Channels for Asterisk

As an open-source project, we would expect Asterisk to have at least some basic community support that we could rely on. Asterisk does have this and it has quite a bit more as well. It has mailing lists, forums, and IRC as well as official support from Digium. We don't always require commercial support but if running Asterisk is not our core responsibility or if we have other constraints, having paid technical support on hand can be a resource we would welcome.

Mailing Lists

There are a few mailing lists available for unofficial Asterisk support, by far the most active being provided by Digium itself. They are frequented by Digium staff as well as Asterisk users and are probably the best source of information when it comes to quick opinions or support from the community. They are found at the following URL:

`http://lists.digium.com/mailman/listinfo`

The USERS mailing list is the best choice for support issues.

There is also the VOIPSEC mailing list provided by `www.voipsa.org`, which isn't Asterisk-centric as its main focus is VoIP security on a wider scale. However as Asterisk is one of the most common VoIP solutions, it is a topic of frequent discussion on the list and topics such as firewalling protocols or encrypted communication are directly relevant to anyone responsible for the security of an Asterisk installation.

We may not decide to use these mailing lists as a support mechanism; however, it is worth having a "lurk" and reading through them at least, to give an insight into how other people are using Asterisk and the problems and issues they come across. Such experiences are invaluable in ensuring we do not repeat others' mistakes and will help in increasing our knowledge of Asterisk and associated technologies. The VOIPSEC list for instance has become the focal point of VoIP security and is often the first outlet for information that has an impact on the security of a VoIP implementation.

Forums

You can also obtain some support from the Digium forums, which can be found at `http://forums.digium.com/`. However, they aren't as busy as the other support available, the mailing lists and IRC being most popular.

IRC (Internet Relay Chat)

Asterisk has a lively community support mechanism provided by its IRC channel. This can be found on the Freenode network, which is a network that comprises almost entirely support channels for free and open source software.

To access this, download a suitable IRC client. mIRC, X-Chat, irssi, and chatzilla are commonly used clients, and most have the address for the Freenode (`irc.freenode.net`) servers in their default configuration. Once connected to Freenode, join `#asterisk`. This channel is much like the Digium mailing list, in that it focuses on discussion of the use and administration of Asterisk. It is also frequented by the same people that use the mailing lists: developers, administrators, and users.

IRC becomes a valid support mechanism when we need quick short answers to our problems or a quick sanity check on something we intend to do. It is often difficult to solve complex problems on IRC especially those that require long detailed explanations or which extend over large portions of our configuration files. It's often easier to explain such matters in an email to a list. However IRC is good for quick replies and for question and answer sessions, so it shouldn't be overlooked.

Digium

Digium provides a variety of services relating to the installation and running of Asterisk, from email and telephone support to on-site support contracts. We should evaluate the need and benefit of this as we decide which kind of support we need. For example if we have a full-time Asterisk administrator or team, we probably wouldn't require much support: maybe email support for occasional troubles, but the mailing lists could possibly provide enough.

If, however, we are employed as a single administrator in an SME we would benefit from having official support mechanisms on hand, although in reality the spread of support is usually the other way around, with SMEs winging it and larger companies having too much support. We also have to consider cost; unofficial community support will obviously be cheaper than paid commercial support. We should evaluate our needs carefully and ensure that we have the necessary support in place to maintain our Asterisk system.

Summary

The phone system of any modern business is something that, if it works well, should be almost invisible to its users. We want them to take it for granted, and to use its features without thinking. It's inevitable and even desirable that our users should come to depend on the services the system offers. Naturally therefore, we want to minimize any disruption to the system, and to make sure that, in the event of a failure, normal service can be resumed as smoothly and quickly as possible.

In this chapter, we've looked at how to be prepared for such an eventuality, by performing regular and systematic backups. We also looked at making a Disaster Recovery Plan, which can help to minimize the time taken to get the system back online.

Of course, the best way to minimize disruption from service outages is to prevent them happening in the first place. To this end, we have looked at how to make Asterisk more robust and how to harden it against attack.

Not all failures are the result of malicious activity, however, and we've also covered a few issues that you should consider in order to make Asterisk scale well. Finally, the community support channels are invaluable in keeping your Asterisk system well maintained and running efficiently, as well as providing help should you ever get stuck. The last section of this chapter was devoted to coverage of these various channels.

[PACKT PUBLISHING]

Index

A

AddQueueMember application, 80
agents
agents.conf, 81
defining in queues.conf, 81
scenario call center, 81
types, 81
agents.conf, 81
AMP
about, 101
configuration options, 104
Flash operator configuration files, 106
Flash operator panel, 105
FOP, 105
maintenance options, 103
announce variable, queues.conf, 64
announce-frequency variable, queues.conf, 65
announce-holdtime variable, queues.conf, 65
Answer action, 72
Asterisk. *See also* PBX
about, 5
agent settings, 81
as IVR, interactive voice response, 8
as PBX, private branch exchange, 5-8
as voicemail system, 8
as VoIP, voice over IP system, 9
Asterisk management portal, 101
backup, 135
cdr_csv module, 92
Comedian Mail, voicemail program, 61
conference rooms, 66
configuration files, 36
configuring, 43
considerations, 12
deployment, planning, 17
directories, 145
extension length, choosing, 29
limitations, 11
maintenance, 135
MP3, streaming files to handset, 63
platform dependency, 12
protocols, supported, 21
queues, 64
reload, 40
restart, 40
sample configuration files, 36
scalability, 152
security, 144
session initiation protocol support, 11
SIP support, 11
starting, 39
support, 154
system maintenance, 135
voicemail, 61
Asterisk deployment
extension length, choosing, 29
hardware requirements, 28
planning, 17
terminal devices, choosing, 25-28
terminal devices, types, 21
Asterisk installation
libpri, 35
prerequisite packages, 33
procedure, 36
source files, 34
Zaptel, 35
Asterisk management portal. *See* AMP
Asterisk security
access control, 145
asterisk.conf, 145
configuration management, 151
DACL, 145
directories, 145
file integrity, checking, 147
permissions, 145
permissions, script, 146
RBAC, 148
root-kits, 147
Tripwire, integrity checker, 147
asterisk.conf, 145
Asterisk@Home. *See also* SugarCRM
about, 99
AMP, 101
Asterisk management portal, 101
CRM, 110

customer relationship management, 110
features, 106, 110
harware requirements, 100
installing, 100
installing, advanced options, 100
PBX, creating, 107
SugarCRM, 110
underlying technology, 99
asterisk_restart.cron script, 142
asterisk-addons distribution, 94
automated attendants
about, 84
configuration file development, 85-87

B

Background action, 72
backup and maintenance
about, 135
areas, 135
asterisk_restart.cron script, 142
backup scripts, 139
backup.cron script, 140
configuration backup, 136, 137
disaster recovery plan, 143
log file backup, 139
monitor_mix.cron script, 141
package management system, 136
schedule, 137
server, restoring, 143
voice data backup, 138
voice data backup, rsync, 138
backup scripts, 139
backup.cron script, 140
Bastille, Asterisk security, 148
bchan, <device> option, 47
BRI, basic rate interface, 18

C

call detail records, *See* CDR (call detail records)
call parking, 82
call queues. *See* queues
calls, handling, 69
calls, monitoring, 95
calls, recording, 96
case study, hosted PBX
configuration, extensions, 131
configuration, music on hold, 130
configuration, sip.conf, 130
configuration, voicemail, 131
configuration, zapata.conf, 130
configuration, zaptel.conf, 129
scenario, 129
case study, small business
configuration, agents.conf, 122
configuration, conference, 124
configuration, extensions, 125, 126
configuration, music on hold, 122
configuration, queues, 122
configuration, sip.conf, 123
configuration, trunkdial, 126
configuration, voicemail, 124
configuration, zapata.conf, 121
configuration, zaptel.conf, 121
planning, 120
scenario, 120
case study, small office/home office
configuration, extensions, 119
configuration, modules, 118
configuration, music on hold, 117
configuration, voicemail, 118
configuration, zapata.conf, 117
configuration, zaptel.conf, 116
planning, 116
scenario, 115
CDR (call detail records), 7, 91
cdr-csv module, 92
flat-file CDR logging, 92
security scenario, 92
cdr_csv module, 92
cdr_pgsql.conf, 94
CentOS, Linux distribution, 99
channels, 47
checksum, 147
chkrootkit tool, 147
Comedian Mail, voicemail program
about, 61
attaching to email, 63
configuring, 61
fast forwarding, 62
format, 61
message length, limiting, 62
rewinding, 62
timezone messages, defining, 62
voicemail box, example, 63
voicemail.conf, 61
communication devices, terminal equipment, 24
conf files
about, 39

sample files, 36
configuration file, samples, 36
context variable, queues.conf, 65
context, creating, 69
CRM. *See* SugarCRM
customer relationship management. *See* SugarCRM

D

DACL, 145
database CDR logging, 93
dchan, <device> option, 47
Dial action, 72
dialplan creation, 69
DID, direct inward dialling, 83
direct inward dialling, 83
discretionary access control lists, 145

E

e&m, <device> option, 47
extension
actions, 72
creating, 71
extension number, 71
extensions.conf, 71
format, 71
outgoing extensions, 75
special extensions, 71
voicemail, transferring to, 73
extension length, 29

F

file integrity, 147
files mode, music on hold, 63
Flash operator panel, 105
flat-file CDR logging, 92
flexibility. *See* Asterisk@Home
FOP, 105
FXO signaling, 48
fxsgs, <device> option, 47
fxsls, <device> option, 47

G

Goto action, 72

H

H.323 protocol, 22
security, 150
handling calls, 69
hard phones, terminal equipment, 21

I

IAX protocol
about, 23
global options, 59
interfaces, 59
security, 151
users, defining, 60
iax.conf, 59
ifconfig, 101
integrated services digital network. *See* ISDN connection
Internet relay chat, 155
ISDN connection
about, 18
BRI, 18
PRI, 18

L

libpri, 35
load balancing, 153

M

mailing lists, 155
maxlen variable, queues.conf, 65
meetme.conf, 66
member variable, queues.conf, 66
monitor_mix.cron script, 141
MP3, streaming files to handset, 63
music variable, queues.conf, 64
musiconhold.conf, 63

N

ntpdate program, 142

O

outgoing extensions, 75, 76, 77

P

parking, 82
parking.conf, 82
PBX
about, 5
Asterisk@Home, 107
call detail records, 7
call distribution, 7
call records, 8
communication devices, 24
hard phone, 21
line trunking, 6
soft phone, 23
station-to-station calls, 6
telco features, 7
phone tree, 84
Playback action, 72
postgres_cdr.sql script, 93
POTS line, connection method, 17
PRI, primary rate interface, 18
private branch exchange. *See* PBX
PSTN
E1 connection method, 18
ISDN (integrated services digital network), 18
POTS (plain old telephone service) line, 17
T1 connection method, 18
VoIP connection, 19
public switched telephony network. *See* PSTN

Q

Queue action, 72
queues
about, 64, 78
application options, 79
call distribution, 78
members, assigning statically, 80
queues.conf, 64, 78
variables, 64
queues.conf, 64

R

RBAC, 148
real-time transport protocol, 151
reload, 40
RemoveQueueMember application, 80
restart, Asterisk
about, 41
options, 40
retry variable, queues.conf, 65
rkhunter tool, 147
role-based access control, 148
root-kits, 147
rsyncd.conf, 137
RTP, 151

S

sample configuration files, 36
scalability, 152
SELinux patch, 148
session initiation protocol, 11
SIP
about, 11, 22
configuring, 54
global options, 54
interfaces, 54
security, 150
users, defining, 56
sip.conf
about, 54
configuration for hosted PBX, 130
cosmall business, 123
soft phones, terminal equipment, 23
spans, 46
strategy variable, queues.conf, 65
su command, 151
sudo, Asterisk access restriction, 152
SugarCRM
about, 110
administration, 112, 113
calls, scheduling, 111
contacts, adding, 111
email, settings, 112
user management, 112
user roles, 113
support, 154

T

T1, defining as span, 46
telephone company (telco), 7
timeout variable, queues.conf, 65
timesync.cron NTP script, 142

Tripwire, integrity checker, 147

U

unused, <device> option, 47

V

Voice over IP. *See* VoIP
Voicemail action, 72, 87
VoicemailMain action, 72
VoIP
Asterisk as, 9

W

WebMeetMe frontend, 106

Z

zapata.conf. *See* Zaptel
Zaptel
channels, configuring, 47, 53
configuring, 44
extensions, 73
global options, 44
installing, 35
interfaces, 44
lines, device class, 45
T1, <device> options, 47
T1, defining as span, 46
terminals, device class, 48
zapata.conf, 48
zapata.conf, lines, 52
zapata.conf, options, 49
zaptel.conf, 44
zaptel.conf, 44

PACKT
PUBLISHING